EXCERPTED FROM

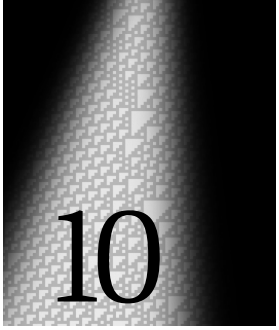
STEPHEN WOLFRAM

# A NEW KIND OF SCIENCE

---

CHAPTER 10

## *Processes of Perception and Analysis*



# 10

# Processes of Perception and Analysis

## Introduction

In the course of the past several chapters, we have discussed the basic mechanisms responsible for a variety of phenomena that occur in nature. But in trying to explain our actual experience of the natural world, we need to consider not only how phenomena are produced in nature, but also how we perceive and analyze these phenomena. For inevitably our experience of the natural world is based in the end not directly on behavior that occurs in nature, but rather on the results of our perception and analysis of this behavior.

Thus, for example, when we look at the behavior of a particular natural system, there will be certain features that we notice with our eyes, and certain features, perhaps different, that we can detect by doing various kinds of mathematical or other analysis.

In previous chapters, I have argued that the basic mechanisms responsible for many processes that occur in nature can be captured by simple computer programs based on simple rules. But what about the processes that are involved in perception and analysis?

Particularly when it comes to the higher levels of perception, there is much that we do not know for certain about this. But what I will argue in this chapter is that the evidence we have suggests that the basic mechanisms at work can once again successfully be captured by simple programs based on simple rules.

In the traditional sciences, it has rarely been thought necessary to discuss in any explicit kind of way the processes that are involved in perception and analysis. For in most cases all that one studies are rather simple features that can readily be extracted by very straightforward processes—and which can for example be described by just a few numbers or by a simple mathematical formula.

But as soon as one tries to investigate behavior of any substantial complexity, the processes of perception and analysis that one needs to use are no longer so straightforward. And the results one gets can then depend on these processes.

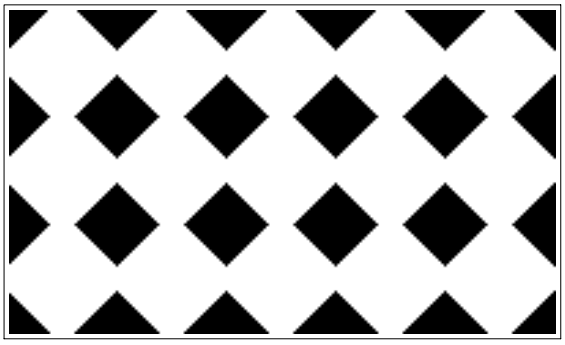
In the traditional sciences it has usually been assumed that any result that is not essentially independent of the processes of perception and analysis used to obtain it cannot be definite or objective enough to be of much scientific value. But the point is that if one explicitly studies processes of perception and analysis, then it becomes possible to make quite definite and objective statements even in such cases.

And indeed some of the most significant conclusions that I will reach at the end of this book are based precisely on comparing the processes that are involved in the production of certain forms of behavior with the processes involved in their perception and analysis.

## What Perception and Analysis Do

In everyday life we are continually bombarded by huge amounts of data, in the form of images, sounds, and so on. To be able to make use of this data we must reduce it to more manageable proportions. And this is what perception and analysis attempt to do. Their role in effect is to take large volumes of raw data and extract from it summaries that we can use.

At the level of raw data the picture at the top of the facing page, for example, can be thought of as consisting of many thousands of individual black and white cells. But with our powers of visual perception and analysis we can immediately see that the picture can be summarized just by saying that it consists essentially of an array of repeated black diamond shapes.



An example of a picture that our powers of perception and analysis readily allow us to summarize quite succinctly in simple geometrical terms. At the lowest level, however, the picture consists of 24,000 black and white cells.

There are in general two ways in which data can be reduced by perception and analysis. First, those aspects of data that are not relevant for whatever purpose one has can simply be ignored. And second, one can avoid explicitly having to specify every element in the data by making use of regularities that one sees.

Thus, for example, in summarizing the picture above, we choose to ignore some details, and then to describe what remains in terms of its simple repetitive overall geometrical structure.

Whenever there are regularities in data, it effectively means that some of the data is redundant. For example, if a particular pattern is repeated, then one need not specify the form of this pattern more than once—for the original data can be reproduced just by repeating a copy of the pattern. And in general, the presence of regularities makes it possible to replace literal descriptions of data by shorter descriptions that are based on procedures for reproducing the data.

There are many forms of perception and analysis. Some happen quite automatically in our eyes, ears and brains—and these we usually call perception. Others require explicit conscious effort and mathematical or computational work—and these we usually call analysis. But the basic goal in all cases is the same: to reduce raw data to a useful summary form.

Such a summary is important whenever one wants to store or communicate data efficiently. It is also important if one wants to compare new data with old, or make meaningful extrapolations or predictions based on data. And in modern information technology the problems of data compression, feature detection, pattern recognition

and system identification all in effect revolve around finding useful summaries of data.

In traditional science statistical analysis has been the most common way of trying to find summaries of data. And in general perception and analysis can be viewed as equivalent to finding models that reproduce whatever aspects of data one considers relevant.

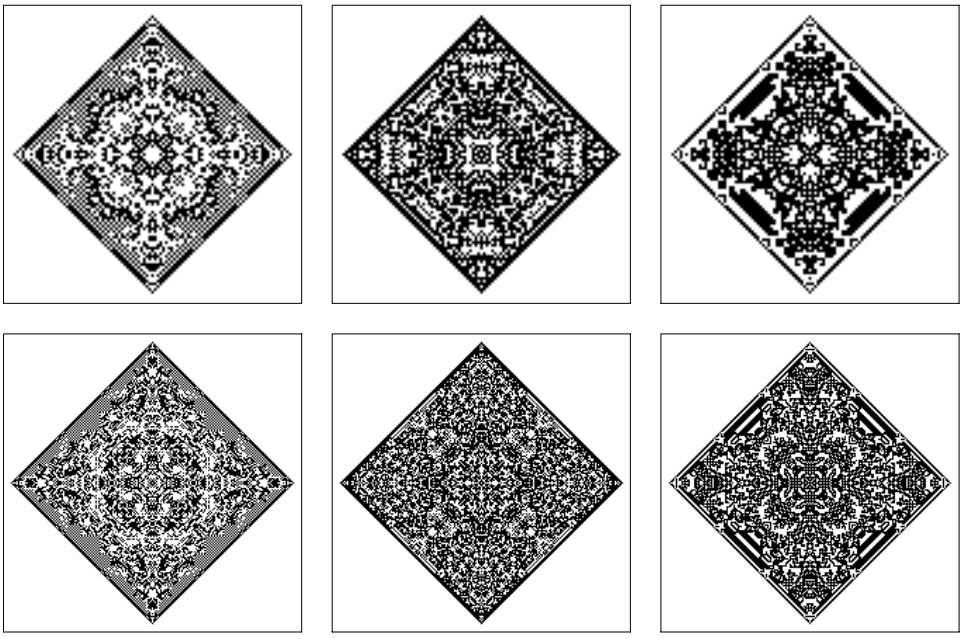
Perception and analysis correspond in many respects to the inverse of most of what we have studied in this book. For typically what we have done is to start from a simple computer program, and then seen what behavior this program produces. But in perception and analysis we start from behavior that we observe, then try to deduce what procedure or program will reproduce this data.

So how easy is it to do this? It turns out that for most of the kinds of rules used in traditional mathematics, it is in fact fairly easy. But for the more general rules that I discuss in this book it appears to often be extremely difficult. For even though the rules may be simple, the behavior they produce is often highly complex, and shows absolutely no obvious trace of its simple origins.

As one example, the pictures on the facing page were all generated by starting from a single black cell and then applying very simple two-dimensional cellular automaton rules. Yet if one looks just at these final pictures, there is no easy way to tell how they were made. Our standard methods of perception and analysis can certainly determine that the pictures are for example symmetrical. But none of these methods typically get even close to being able to recognize just how simple a procedure can in fact be used to produce the pictures.

One might think that our inability to find such a procedure could just be a consequence of limitations in the particular methods of perception and analysis that we, as humans, happen to have developed. And one might therefore suppose that an alien intelligence could exist which would be able to look at our pictures and immediately tell that they were produced by a very simple procedure.

But in fact I very much doubt that this will ever be the case. For I suspect that there are fundamental limitations on what perception and analysis can ever be expected to do. For there seem to be many kinds of



Patterns produced by taking a single black cell, then evolving for 50 and 100 steps according to outer totalistic cellular automaton rules 54, 222 and 374. Despite the simple description that can be given of this procedure, our standard methods of perception and analysis cannot readily deduce this description given just the final pictures shown here.

systems in which it is overwhelmingly easier to generate highly complex behavior than to recognize the origins of this behavior.

As I have discovered in this book, it is rather easy to generate complex behavior by starting from simple initial conditions and then following simple sets of rules. But the point is that if one starts from some particular piece of behavior there are in general no such simple rules that allow one to go backwards and find out how this behavior can be produced. Typically the problem is similar to trying to find solutions that will satisfy certain constraints. And as we have seen several times in this book, such problems can be extremely difficult.

So insofar as the actual processes of perception and analysis that end up being used are fairly simple, it is inevitable that there will be situations where one cannot recognize the origins of behavior that one sees—even when this behavior is in fact produced by very simple rules.

## Defining the Notion of Randomness

Many times in this book I have said that the behavior of some system or another seems random. But so far I have given no precise definition of what I mean by randomness. And what we will discover in this section is that to come up with an appropriate definition one has no choice but to consider issues of perception and analysis.

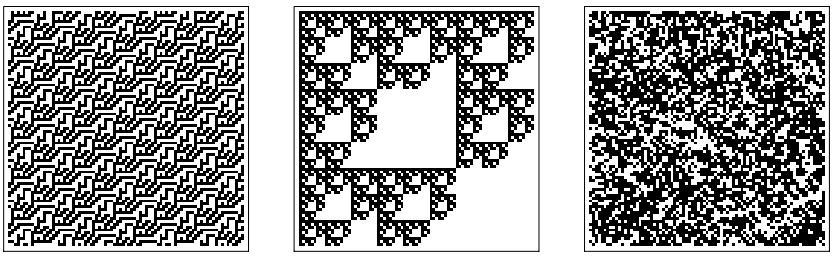
One might have thought that from traditional mathematics and statistics there would long ago have emerged some standard definition of randomness. But despite occasional claims for particular definitions, the concept of randomness has in fact remained quite obscure. And indeed I believe that it is only with the discoveries in this book that one is finally now in a position to develop a real understanding of what randomness is.

At the level of everyday language, when we say that something seems random what we usually mean is that there are no significant regularities in it that we can discern—at least with whatever methods of perception and analysis we use.

We would not usually say, therefore, that either of the first two pictures at the top of the facing page seem random, since we can readily recognize highly regular repetitive and nested patterns in them. But the third picture we would probably say does seem random, since at least at the level of ordinary visual perception we cannot recognize any significant regularities in it.

So given this everyday notion of randomness, how can we build on it to develop more precise definitions? The first step is to clarify what it means not to be able to recognize regularities in something. Following the discussion in the previous section, we know that whenever we find regularities, it implies that redundancy is present, and this in turn means that a shorter description can be given. So when we say that we cannot recognize any regularities, this is equivalent to saying that we cannot find a shorter description.

The three pictures on the facing page can always be described by explicitly giving a list of the colors of each of the 6561 cells that they contain. But by using the regularities that we can see in the first two



(a) (b) (c)

Pictures exhibiting different degrees of apparent randomness. Pictures (a) and (b) have obvious regularities, and would never be considered particularly random. But picture (c) has almost no obvious regularities, and would typically be considered quite random. As it turns out, picture (c), like (a) and (b), can actually be generated by a quite simple process. But the point is that the simplicity of this process does not affect the fact that with our standard methods of perception and analysis picture (c) is for practical purposes random.

pictures, we can readily construct much shorter—yet still complete—descriptions of these pictures.

The repetitive structure of picture (a) implies that to reproduce this picture all we need do is to specify the colors in a $49 \times 2$ block, and then say that this block should be repeated an appropriate number of times. Similarly, the nested structure of picture (b) implies that to reproduce this picture, all we need do is to specify the colors in a $3 \times 3$ block, and then say that as in a two-dimensional substitution system each black cell should repeatedly be replaced by this block.

But what about picture (c)? Is there any short description that can be given of this picture? Or do we have no choice but just to specify explicitly the color of every one of the cells it contains?

Our powers of visual perception certainly do not reveal any significant regularities that would allow us to construct a shorter description. And neither, it turns out, do any standard methods of mathematical or statistical analysis. And so for practical purposes we have little choice but just to specify explicitly the color of each cell.

But the fact that no short description can be found by our usual processes of perception and analysis does not in any sense mean that no such description exists at all. And indeed, as it happens, picture (c) in fact allows a very short description. For it can be generated just by

starting with a single black cell and then applying a simple two-dimensional cellular automaton rule 250 times.

But does the existence of this short description mean that picture (c) should not be considered random? From a practical point of view the fact that a short description may exist is presumably not too relevant if we can never find this description by any of the methods of perception and analysis that are available to us. But from a conceptual point of view it may seem unsatisfactory to have a definition of randomness that depends on our methods of perception and analysis, and is not somehow absolute.

So one possibility is to define randomness so that something is considered random only if no short description whatsoever exists of it. And before the discoveries in this book such a definition might have seemed not far from our everyday notion of randomness. For we would probably have assumed that anything generated from a sufficiently short description would necessarily look fairly simple. But what we have discovered in this book is that this is absolutely not the case, and that in fact even from rules with very short descriptions it is easy to generate behavior in which our standard methods of perception and analysis recognize no significant regularities.

So to say that something is random only if no short description whatsoever exists of it turns out to be a highly restrictive definition of randomness. And in fact, as I mentioned in Chapter 7, it essentially implies that no process based on definite rules can ever manage to generate randomness when there is no randomness before. For since the rules themselves have a short description, anything generated by following them will also have a correspondingly short description, and will therefore not be considered random according to this definition.

And even if one is not concerned about where randomness might come from, there is still a further problem: it turns out in general to be impossible to determine in any finite way whether any particular thing can ever be generated from a short description. One might imagine that one could always just try running all programs with progressively longer descriptions, and see whether any of them ever generate what one wants. But the problem is that one can never in general tell in

advance how many steps of evolution one will need to look at in order to be sure that any particular piece of behavior will not occur. And as a result, no finite process can in general be used to guarantee that there is no short description that exists of a particular thing.

By setting up various restrictions, say on the number of steps of evolution that will be allowed, it is possible to obtain slightly more tractable definitions of randomness. But even in such cases the amount of computational work required to determine whether something should be considered random is typically astronomically large. And more important, while such definitions may perhaps be of some conceptual interest, they correspond very poorly with our intuitive notion of randomness. In fact, if one followed such a definition most of the pictures in this book that I have said look random—including for example picture (c) on page 553—would be considered not random. And following the discussion of Chapter 7, so would at least many of the phenomena in nature that we normally think of as random.

Indeed, what I suspect is that ultimately no useful definition of randomness can be based solely on the issue of what short descriptions of something may in principle exist. Rather, any useful definition must, I believe, make at least some reference to how such short descriptions are supposed to be found.

Over the years, a variety of definitions of randomness have been proposed that are based on the absence of certain specific regularities. Often these definitions are presented as somehow being fundamental. But in fact they typically correspond just to seeing whether some particular process—and usually a rather simple one—succeeds in recognizing regularities and thus in generating a shorter description.

A common example—to be discussed further two sections from now—involves taking, say, a sequence of black and white cells, and then counting the frequency with which each color and each block of colors occurs. Any deviation from equality among these frequencies represents a regularity in the sequence and reveals nonrandomness. But despite some confusion in the past it is certainly not true that just checking equality of frequencies of blocks of colors—even arbitrarily long ones—is sufficient to ensure that no regularities at all exist. This

procedure can indeed be used to check that no purely repetitive pattern exists, but as we will see later in this chapter, it does not successfully detect the presence of even certain highly regular nested patterns.

So how then can we develop a useful yet precise definition of randomness? What we need is essentially just a precise version of the statement at the beginning of this section: that something should be considered random if none of our standard methods of perception and analysis succeed in detecting any regularities in it. But how can we ever expect to find any kind of precise general characterization of what all our various standard methods of perception and analysis do?

The key point that will emerge in this chapter is that in the end essentially all these methods can be viewed as being based on rather simple programs. So this suggests a definition that can be given of randomness: something should be considered to be random whenever there is essentially no simple program that can succeed in detecting regularities in it.

Usually if what one is studying was itself created by a simple program then there will be a few closely related programs that always succeed in detecting regularities. But if something can reasonably be considered random, then the point is that the vast majority of simple programs should not be able to detect any regularities in it.

So does one really need to try essentially all sufficiently simple programs in order to determine this? In my experience, the answer tends to be no. For once a few simple programs corresponding to a few standard methods of perception and analysis have failed to detect regularities, it is extremely rare for any other simple program to succeed in detecting them.

So this means that the everyday definition of randomness that we discussed at the very beginning of this section is in the end already quite unambiguous. For it typically will not matter much which of the standard methods of perception and analysis we use: after trying a few of them we will almost always be in a position to come to a quite definite conclusion about whether or not something should be considered random.

## Defining Complexity

Much of what I have done in this book has been concerned in one way or another with phenomena associated with complexity. But just as one does not need a formal definition of life in order to study biology, so also it has not turned out to be necessary so far in this book to have a formal definition of complexity. Nevertheless, following our discussion of randomness in the previous section, we are now in a position to consider how the notion of complexity might be formally defined.

In everyday language, when we say that something seems complex what we typically mean is that we have not managed to find any simple description of it—or at least of those features of it in which we happen to be interested. But the goal of perception and analysis is precisely to find such descriptions, so when we say that something seems complex, what we are effectively saying is that our powers of perception and analysis have failed on it.

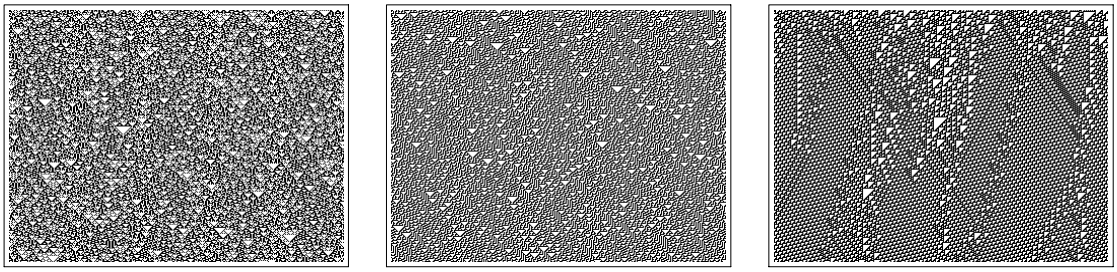
As we discussed two sections ago, there are two ways in which perception and analysis can typically operate. First, they can just throw away details in which we are not interested. And second, they can remove redundancy that is associated with any regularities that they manage to recognize.

The definition of randomness that we discussed in the previous section was based on the failure of the second of these two functions. For what it said was that something should be considered random if our standard methods of perception and analysis could not find any short description from which the thing could faithfully be reproduced.

But in defining complexity we need to consider both functions of perception and analysis. For what we want to know is not whether a simple or short description can be found of every detail of something, but merely whether such a description can be found of those features in which we happen to be interested.

In everyday language, the terms "complexity" and "randomness" are sometimes used almost interchangeably. And for example any of the three pictures at the top of the next page could potentially be referred to as either "quite random" or "quite complex". But if one chooses to look

only at overall features, then typically one would tend to say that the third picture seems more complex than the other two.




Examples of pictures that at an everyday level one might typically describe either as being "quite random" or as being "quite complex".

For even though the detailed placement of black and white cells in the first two pictures does not seem simple to describe, at an overall level these pictures still admit a quite simple description: in essence they just involve a kind of uniform randomness in which every region looks more or less the same as every other. But the third picture shows no such uniformity, even at an overall level. And as a result, we cannot give a short description of it even if we ignore its small-scale details.

Of course, if one goes to an extreme and looks, say, only at how big each picture is, then all three pictures have very short descriptions. And in general how short a description of something one can find will depend on what features of it one wants to capture—which is why one may end up ascribing a different complexity to something when one looks at it for different purposes.

But if one uses a particular method of perception or analysis, then one can always see how short a description this manages to produce. And the shorter the description is, the lower one considers the complexity to be.

But to what extent is it possible to define a notion of complexity that is independent of the details of specific methods of perception and analysis? In this chapter I argue that essentially all common forms of perception and analysis correspond to rather simple programs. And if one is interested in descriptions in which no information is lost—as in the discussion of randomness in the previous section—then as I

mentioned in the previous section, it seems in practice that different simple programs usually agree quite well in their ability or inability to find short descriptions.

But this seems to be considerably less true when one is dealing with descriptions in which information can be lost. For it is rather common to see cases in which only a few features of a system may be difficult to describe—and depending on whether or not a given program happens to be sensitive to these features it can ascribe either a quite high or a quite low complexity to the system.

Nevertheless, as a practical matter, by far the most common way in which we determine levels of complexity is by using our eyes and our powers of visual perception. So in practice what we most often mean when we say that something seems complex is that the particular processes that are involved in human visual perception have failed to extract a short description.

And indeed I suspect that even below the level of conscious thought our brains already have a rather definite notion of complexity. For when we are presented with a complex image, our eyes tend to dwell on it, presumably in an effort to give our brains a chance to extract a simple description.

If we can find no simple features whatsoever—as in the case of perfect randomness—then we tend to lose interest. But somehow the images that draw us in the most—and typically that we find the most aesthetically pleasing—are those for which some features are simple for us to describe, but others have no short description that can be found by any of our standard processes of visual perception.

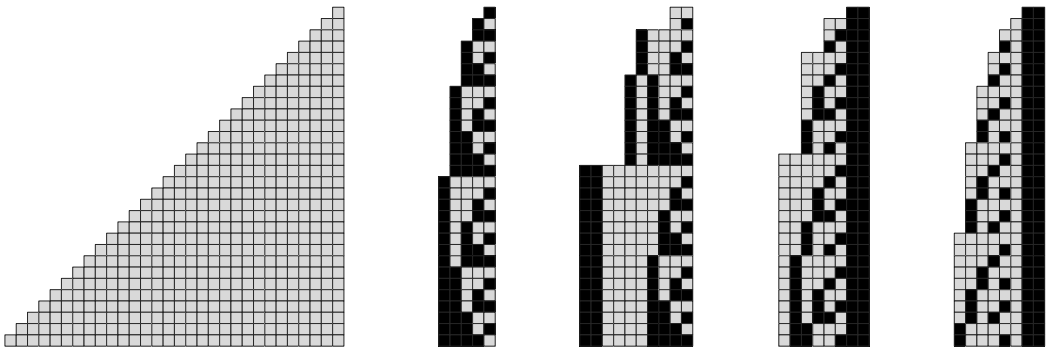
Before the discoveries in this book, one might have thought that to create anything with a significant level of apparent complexity would necessarily require a procedure which itself had significant complexity. But what we have discovered in this book is that in fact there are remarkably simple programs that produce behavior of great complexity. And what this means—as the images in this book repeatedly demonstrate—is that in the end it is rather easy to make pictures for which our visual system can find no simple overall description.

## Data Compression

One usually thinks of perception and analysis as being done mainly in order to provide material for direct human consumption. But in most modern computer and communications systems there are processes equivalent to perception and analysis that happen all the time when data is compressed for more efficient storage or transmission.

One simple example of such a process is run-length encoding—a method widely used in practice to compress data that involves long sequences of identical elements, such as bitmap images of pages of text with large areas of white.

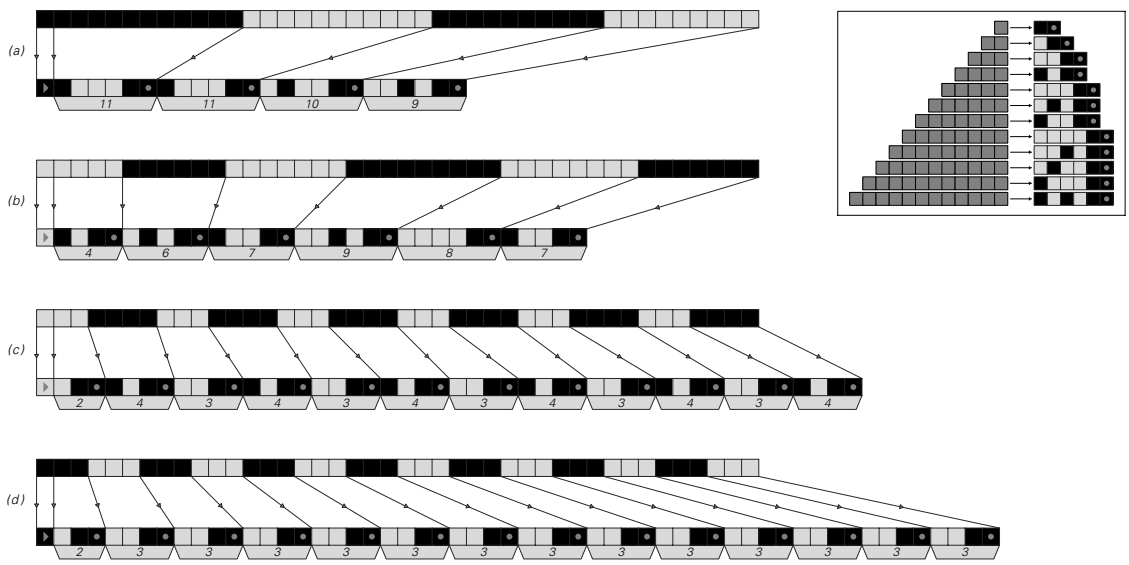
The basic idea of run-length encoding is to break data into runs of identical elements, and then to specify the data just by giving the lengths of these runs. This means, for example, that instead of having to list explicitly all the cells in a run of, say, 53 identical cells, one instead just gives the number "53". And the point is that even if the "53" is itself represented in terms of black and white cells, this representation can be much shorter than 53 cells.




Various representations of numbers from 1 to 30. (a) is unary, in which any given number is represented by a sequence of cells whose length is equal to that number. (b) is ordinary binary or base 2 representation. (c), (d) and (e) are set up to be self-delimiting, so that the end of a number can be recognized purely by looking at the cells within it. (c) is like (b), except that it has a specification of the number of digits at the front. (d) is essentially binary-coded-ternary, with the end of the number indicated by a pair of black cells. (e) uses a non-integer base derived from the Fibonacci sequence, with the property that a pair of black cells can appear only at the end of each number.

Indeed, any digit sequence can be thought of as providing a short representation for a number. But for run-length encoding it turns out that ordinary base 2 digit sequences do not quite work. For if the numbers corresponding to the lengths of successive runs are given one after another then there is no way to tell where the digits of one number end and the next begin.

Several approaches can be used, however, to avoid this problem. One, illustrated in picture (c) at the bottom of the facing page, is to insert at the beginning of each number a specification of how many digits the number contains. Another approach, illustrated in picture (d), is in effect to have two cells representing each digit, and then to indicate the end of the number by a pair of black cells. A variant on this approach, illustrated in picture (e), uses a non-integer base in which pairs of black cells can occur only at the end of a number.

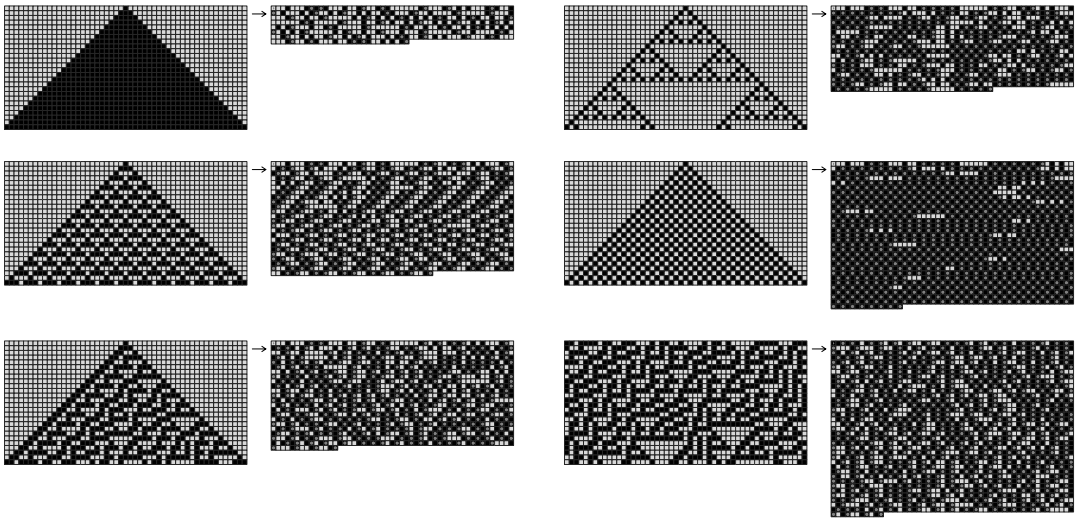



Examples of run-length encoding. In each case the input data is shown on top, and the output is shown below. The arrows between input and output show how the data is broken into runs of identical elements. Each run is then specified by a number, represented as a sequence ending with two black cells, as indicated in the inset picture, and in picture (e) on the facing page. For the first two sets of input data there are enough long runs present that compression is achieved. But for the other two sets no compression is achieved. Note that the first cell in the output is used to specify whether the first run is black or white. In this picture and those that follow, the output consists purely of black and white cells; the gray annotations are included purely as aids to interpretation.

For small numbers, all these approaches yield representations that are at least somewhat longer than the explicit sequences shown in picture (a). But for larger numbers, the representations quickly become much shorter. And this means that they can potentially be used to achieve compression in run-length encoding.

The pictures at the bottom of the previous page show what happens when one applies run-length encoding using representation (e) from page 560 to various sequences of data. In the first two cases, there are sufficiently many long runs that compression is achieved. But in the last two cases, there are too many short runs, and the output from run-length encoding is actually longer than the input.

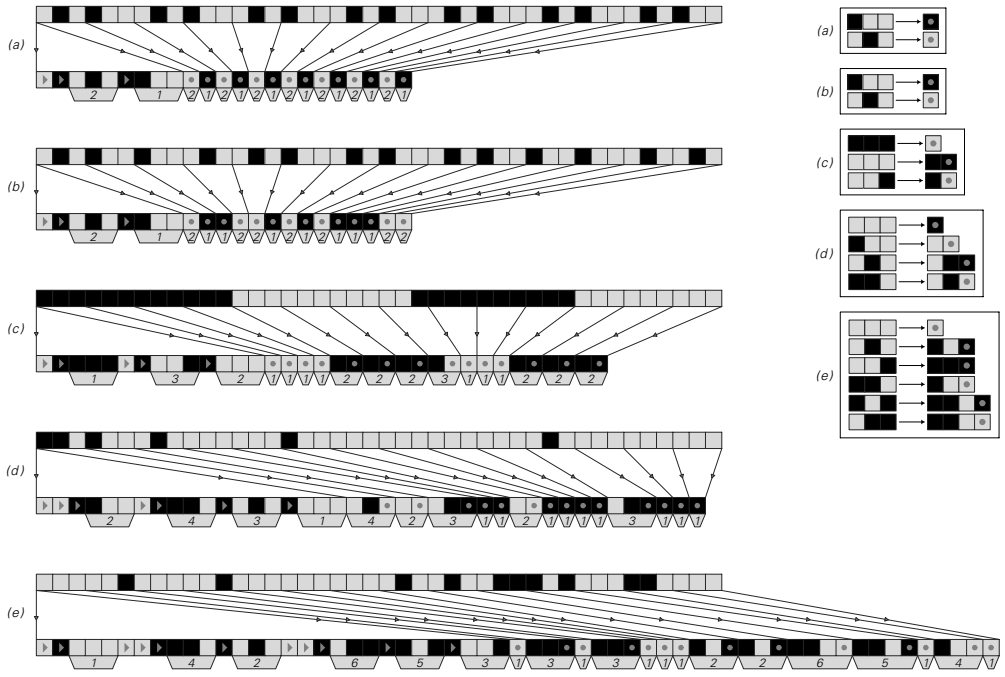
The pictures below show the results of applying run-length encoding to typical patterns produced by cellular automata. When the patterns contain enough regions of uniform color, compression is achieved. But when the patterns are more intricate—even in a simple repetitive way—little or no compression is achieved.



Examples of applying run-length encoding to patterns produced by cellular automata. Successive rows in each original image are placed end to end so as to give a one-dimensional sequence, then run-length encoded, and then chopped into rows again. Compression is typically achieved whenever most of the image consists of regions of uniform color.

Run-length encoding is based on the idea of breaking data up into runs of identical elements of varying lengths. Another common approach to data compression is based on forming blocks of fixed length, and then representing whatever distinct blocks occur by specific codewords.

The pictures below show a few examples of how this works. In each case the input is taken to be broken into blocks of length 3. In the first two cases, there are then only two distinct blocks that occur, so each of these can be represented by a codeword consisting of a single cell, with the result that substantial compression is achieved.

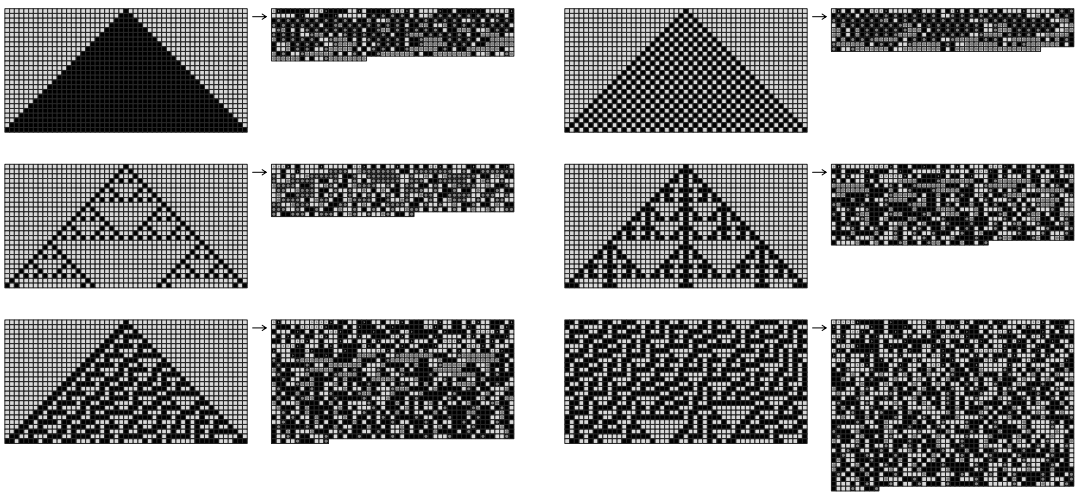



Examples of Huffman coding based on blocks of length 3. In cases (a) and (b), only two possible blocks occur, and these are assigned codewords consisting of a single black cell and a single white cell. In case (c), 3 possible blocks occur; the most common is assigned a codeword consisting of a single white cell, while the others are assigned codewords consisting of two cells. In case (d) 4 out of the 8 possible blocks occur, while in case (e) 6 occur. In all cases, the output begins with a preamble specifying which block is to be represented by which codeword. The blocks appear explicitly in this preamble, and are indicated by numbered tabs. The codewords are represented implicitly by the arrangement of cells shown with arrows. The preamble is followed by the actual codewords representing the data. The codewords are self-delimiting, so that they can be given one after another, with no separator in between.

When a larger number of distinct blocks occur, longer codewords must inevitably be used. But compression can still be achieved if the codewords for common blocks are sufficiently much shorter than the blocks themselves.

One simple strategy for assigning codewords is to number all distinct blocks in order of decreasing frequency, and then just to use the resulting numbers—given, say, in one of the representations discussed above—as the codewords. But if one takes into account the actual frequencies of different blocks, as well as their ranking, then it turns out that there are better ways to assign codewords.

The pictures below show examples based on a method known as Huffman coding. In each case the first part of the output specifies which blocks are to be represented by which codewords, and then the remainder of the output gives the actual succession of codewords that correspond to the blocks appearing in the data. And as the pictures below illustrate, whenever there are fairly few distinct blocks that occur with high frequency, substantial compression is achieved.

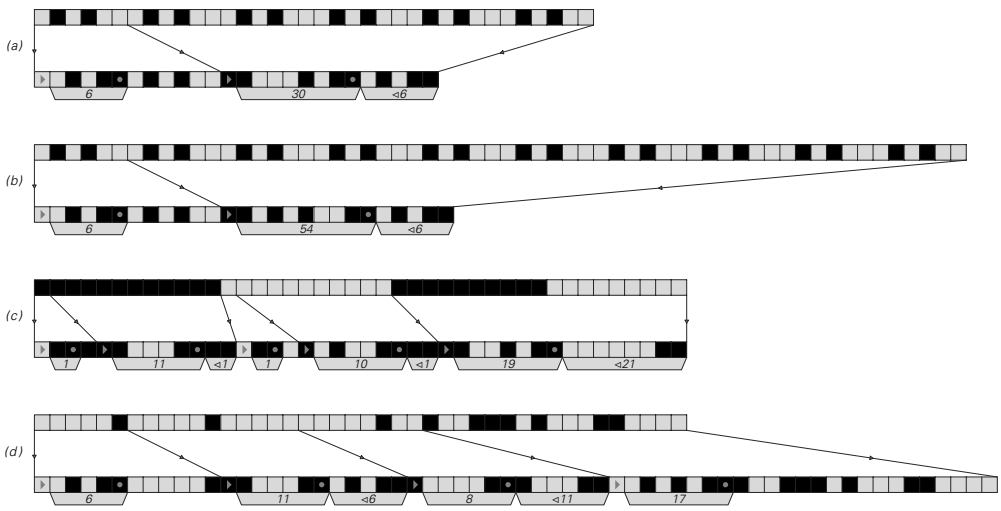


Huffman encoding with blocks of length 6 applied to patterns produced by cellular automata. The maximum possible compression is by a factor of 6; the maximum achieved here is roughly a factor of 3. The difference between the size of the results for the last two examples is mostly a consequence of the presence of large areas of white in the first of them.

But ultimately there is a limit to the degree of compression that can be obtained with this method. For even in the very best case any block of cells in the input can never be compressed to less than one cell in the output.

So how can one achieve greater compression? One approach—which turns out to be similar to what is used in practice in most current high-performance general-purpose compression systems—is to set up an encoding in which any particular sequence of elements above some length is given explicitly only once, and all subsequent occurrences of the same sequence are specified by pointers back to the first one.

The pictures below show what happens when this approach is applied to a few short sequences. In each case, the output consists of two kinds of objects, one giving sequences that are occurring for the first time, and the other giving pointers to sequences that have occurred before. Both kinds of objects start with a single cell that specifies their type. This is

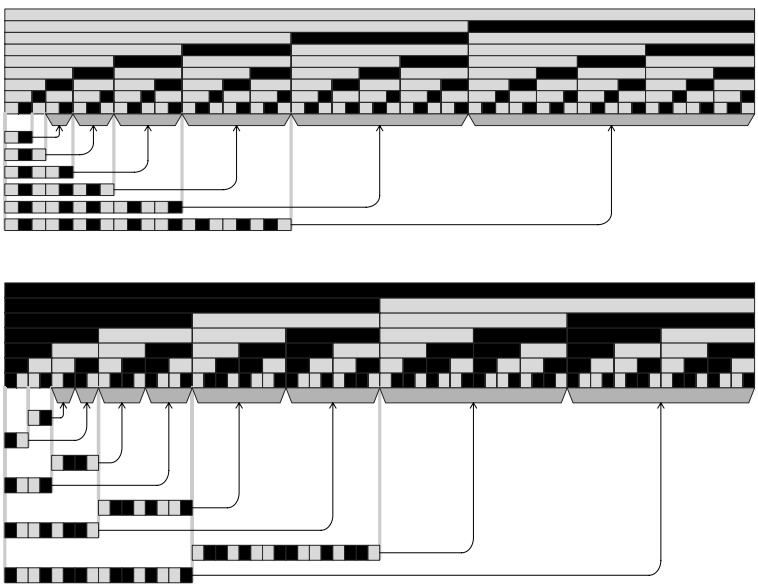



Examples of pointer-based encoding, in which sequences that have occurred once in the data are subsequently specified just by pointers. Each section of output starts with an element which indicates whether what follows is a new sequence, or a pointer to a previous one. After this comes a specification of the length of sequence represented by this section of output, with the number given in the form used for run-length encoding above. Then comes either a literal sequence, or a number giving the offset to where the required sequence last occurred in the data. In the examples shown, pointers are used only for sequences of length at least 6. Pointer-based encoding is similar to the Lempel-Ziv algorithm widely used in practical high-performance general-purpose compression systems.

followed by a specification of the length of the sequence that the object describes. In the first kind of object, the actual sequence is then given, while in the second kind of object what is given is a specification of how far back in the data the required sequence can be found.

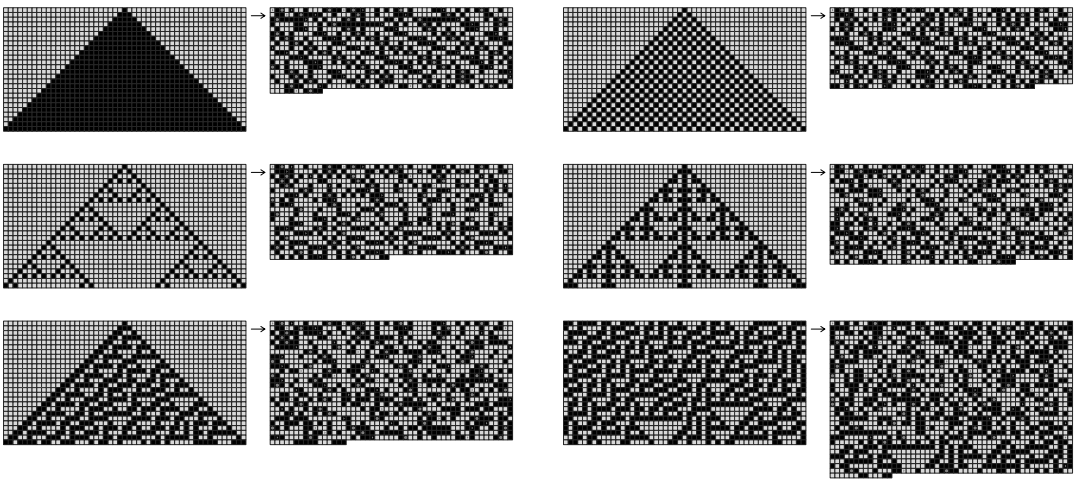
With data that is purely repetitive this method achieves quite dramatic compression. For having once specified the basic sequence to be repeated, all that then needs to be given is a pointer to this sequence, together with a representation of the total length of the data.

Purely nested data can also be compressed nearly as much. For as the pictures below illustrate, each whole level of nesting can be viewed just as adding a fixed number of repeated sequences.



Examples of the pattern of repeats found in purely nested data. As indicated in these pictures, any such data must correspond to the output of a neighbor-independent substitution system (see page 83). In pointer-based encoding, the number of pointers required to represent the data increases essentially like the number of steps in the evolution of the substitution system. Taking into account the length of the representation for each pointer, the compressed form of a nested sequence of length $n$ will typically grow in length like $Log[n]^2$. (This can be compared with $Log[n]$ growth for a purely repetitive sequence.) Note that actual algorithms for pointer-based encoding will typically find a slightly less regular pattern of repeats than is shown in the pictures here.

So what about two-dimensional patterns? The pictures below show what happens if one takes various patterns, arranges their rows one after another in a long line, and then applies pointer-based encoding to the resulting sequences. When there are obvious regularities in the original pattern, some compression is normally achieved—but in most cases the amount is not spectacular.

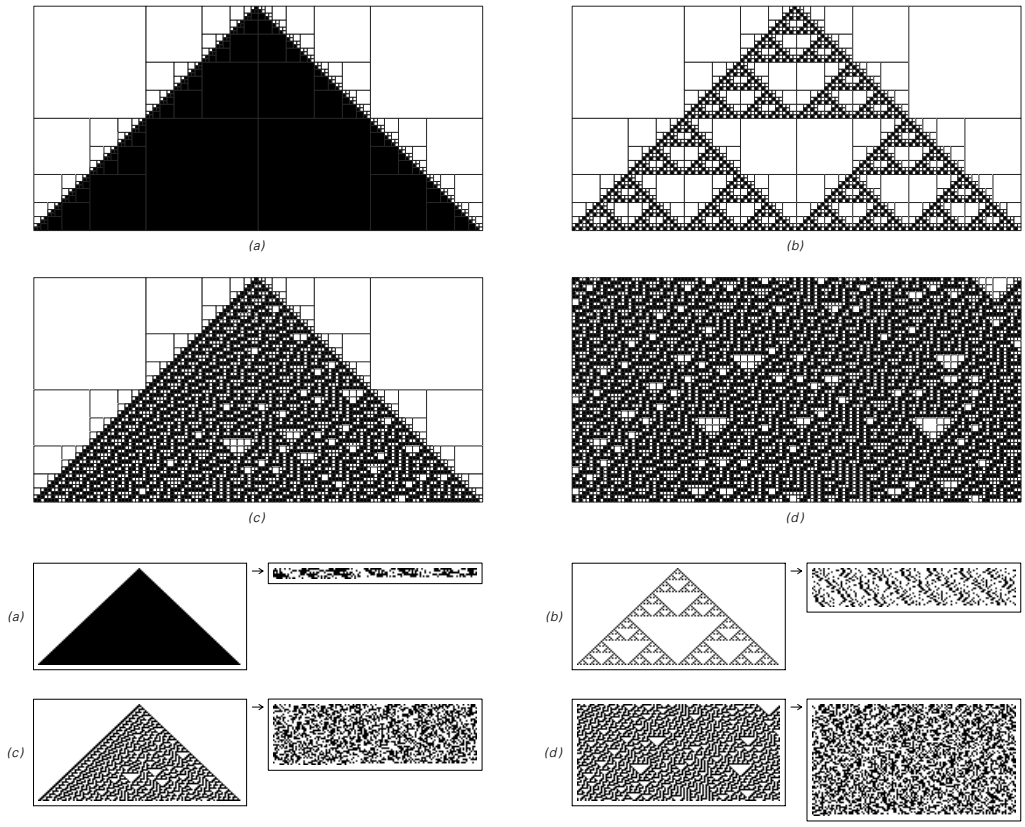


Examples of one-dimensional pointer-based encoding applied to patterns produced by cellular automata. Successive rows in each image are placed end to end so as to get a sequence to which the encoding can be applied. Pointers are used only for repeats that are of length at least 4. In the last example, large regions contain no such repeats, and therefore appear in the output just as they do in the input.

So how can one do better? The basic answer is that one needs to take account of the two-dimensional nature of the patterns. Most compression schemes used in practice have in the past primarily been set up just to handle one-dimensional sequences. But it is not difficult to set up schemes that operate directly on two-dimensional data.

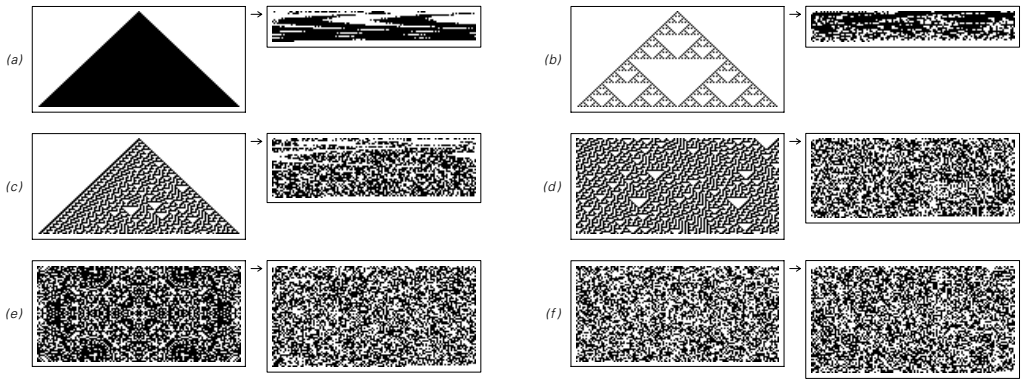
The picture on the next page shows one approach based on the idea of breaking images up into collections of nested pieces, each with a uniform color. In some respects this scheme is a two-dimensional analog of run-length encoding, and when there are large regions of uniform color it yields significant compression.

It is also easy to extend block-based encoding to two dimensions: all one need do is to assign codewords to two-dimensional rather than



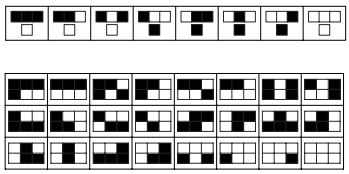

Examples of encoding by two-dimensional recursive subdivision. The idea is to use a generalization of a two-dimensional substitution system, in which at each step a square either remains the same or is subdivided into four small squares. The encoding specifies which choice is made at each step for each square. The method is analogous to the quadtree representation sometimes used in computer graphics. The substantial compression seen even in case (c) is a consequence of the large areas of uniform white that are present.

one-dimensional blocks. And as the pictures at the top of the facing page demonstrate, this procedure can lead to substantial compression. Particularly notable is what happens in case (d). For even though this pattern is produced by a simple one-dimensional cellular automaton rule, and even though one can see by eye that it contains at least some small-scale regularities, none of the schemes we have discussed up till now have succeeded in compressing it at all.




Examples of two-dimensional block-based encoding. Each image is broken into 3 × 2 blocks, and codewords are then assigned to these blocks using the Huffman scheme. Note the presence of compression even in case (d). This is a consequence of the fact that the cellular automaton rule allows only certain blocks to appear in the pattern, as illustrated in the picture below. (e) is generated by a two-dimensional cellular automaton; (f) is the sequence that appears on the center column of rule 30.

The picture below demonstrates why two-dimensional block encoding does, however, manage to compress it. The point is that the two-dimensional blocks that one forms always contain cells whose colors are connected by the cellular automaton rule—and this greatly reduces the number of different arrangements of colors that can occur.



Cellular automaton rule 30, and the 3 × 2 blocks which appear in large patterns generated by it. There are a total of $2^6 = 64$ possible 3 × 2 blocks of black and white cells; the fact that only 24 of them appear in patterns generated by rule 30 is what makes it possible for two-dimensional block-based encoding to compress such patterns.

In cases (e) and (f), however, there is no simple rule for going from one row to the next, and two-dimensional block encoding—like all the other encoding schemes we have discussed so far—does not yield any substantial compression.

Like block encoding, pointer-based encoding can also be extended to two dimensions. The basic idea is just to scan two-dimensional data looking for repeats not of one-dimensional sequences, but instead of two-dimensional regions. And although such a procedure does not in the

past appear to have been used in practice, it is quite straightforward to implement. The pictures on the facing page show some examples of the results one gets. And in many cases it turns out that the overall level of compression obtained is considerably greater than with any of the other schemes discussed in this section. But what is perhaps still more striking is that the patterns of repeated regions seem to capture almost every regularity that we readily notice by eye—as well as some that we do not. In pictures (c) and (d), for example, fairly subtle repetition on the left-hand side is captured, as is fourfold symmetry in picture (e).

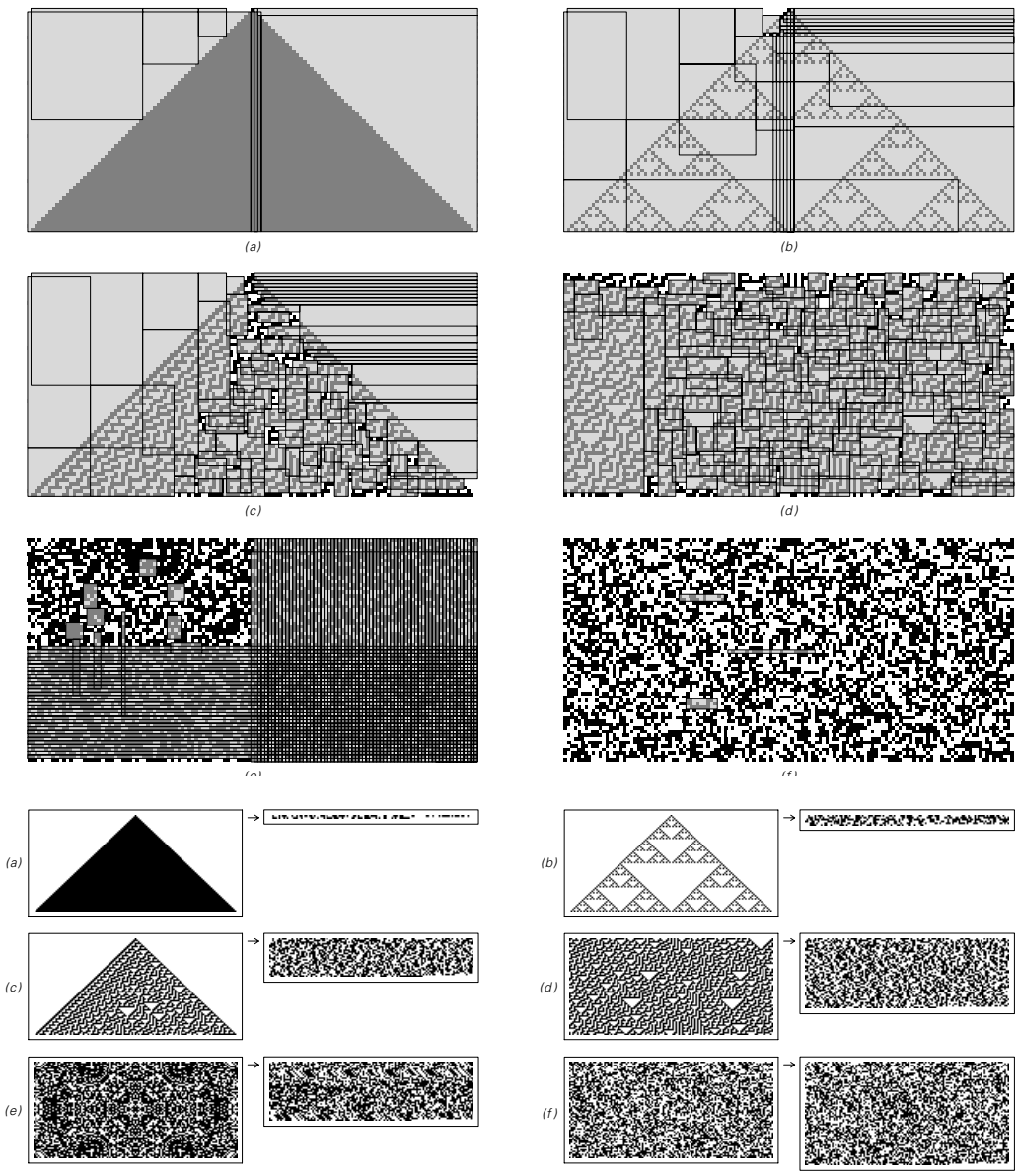
One might have thought that to capture all these kinds of regularities would require a whole collection of complicated procedures. But what the pictures on the facing page demonstrate is that in fact just a single rather straightforward procedure is quite sufficient. And indeed the amount of compression achieved by this procedure in different cases seems to agree rather well with our intuitive impression of how much regularity is present.

All of the methods of data compression that we have discussed in this section can be thought of as corresponding to fairly simple programs. But each method involves a program with a rather different structure, and so one might think that it would inevitably be sensitive to rather different kinds of regularities.

But what we have seen in this section is that in fact different methods of data compression have remarkably similar characteristics. Essentially every method, for example, will successfully compress large regions of uniform color. And most methods manage to compress behavior that is repetitive, and at least to some extent behavior that is nested—exactly the two kinds of simple behavior that we have noted many times in this book.

For more complicated behavior, however, none of the methods seem capable of substantial compression. It is not that no compression is ever in principle possible. Indeed, as it happens, every single one of the pictures on the facing page can for example be generated from very short cellular automaton programs.

But the point is that except when the overall behavior shows repetition or nesting none of the standard methods of data compression



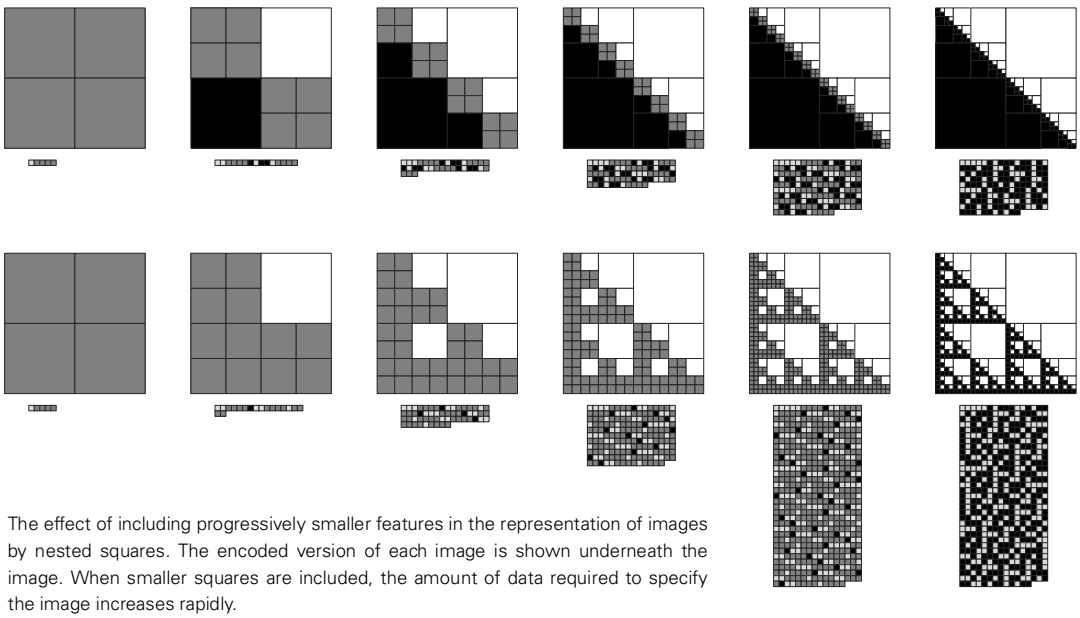

Examples of two-dimensional pointer-based encoding. The gray rectangles in the upper pictures indicate repeated regions that are encoded using pointers. In the particular scheme used here, each of these regions is required to contain at least 25 cells that have not already been encoded using pointers. The images are scanned sequentially and at every point the maximal rectangle extending to the right and down is found that is a repeat of a rectangle previously encountered, and contains the largest number of cells not already encoded using pointers. In many cases this maximal rectangle overlaps those found at subsequent points.

as we have discussed them in this section come even close to finding such short descriptions. And as a result, at least with respect to any of these methods all we can reasonably say is that the behavior we see seems for practical purposes random.

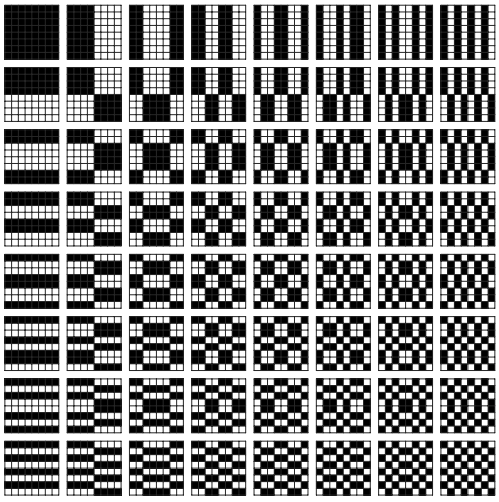
## Irreversible Data Compression

All the methods of data compression that we discussed in the previous section are set up to be reversible, in the sense that from the encoded version of any piece of data it is always possible to recover every detail of the original. And if one is dealing with data that corresponds to text or programs such reversibility is typically essential. But with images or sounds it is typically no longer so necessary: for in such cases all that in the end usually matters is that one be able to recover something that looks or sounds right. And by being able to drop details that have little or no perceptible effect one can often achieve much higher levels of compression.

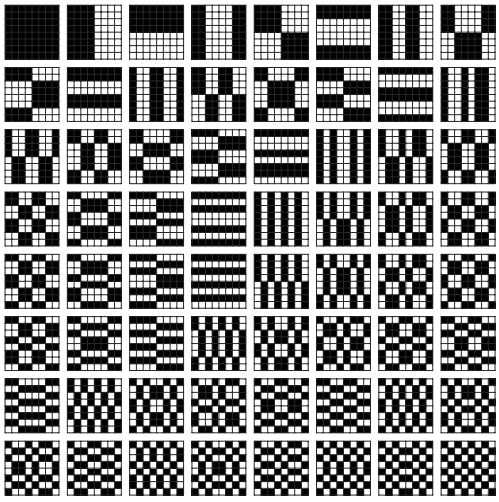
In the case of images a simple approach is just to ignore features that are smaller than some minimum size. The pictures below show



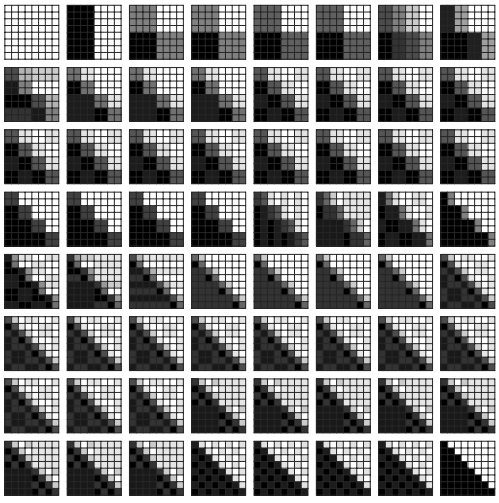
The effect of including progressively smaller features in the representation of images by nested squares. The encoded version of each image is shown underneath the image. When smaller squares are included, the amount of data required to specify the image increases rapidly.



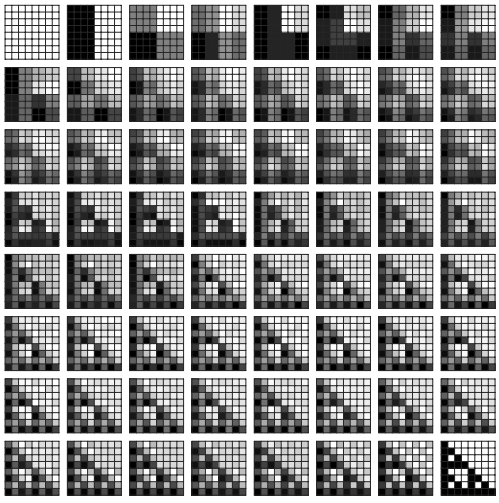
basic forms



ranked basic forms



(a)



(b)

Examples of how images can be built up by adding together basic forms consisting of so-called two-dimensional Walsh functions. On the top left the basic forms are given in so-called sequency order. On the top right they are reordered roughly so as to go systematically from coarser to finer. In the bottom arrays of pictures each successive picture is obtained by adding in the corresponding basic form with an appropriate weight. The basic forms shown here have the property of being orthogonal, so that the weight for each form can be deduced simply by multiplying the original image by that form. Note that the forms involve numerical values -1 and +1, corresponding to cells colored white and black. The images shown here are all rescaled so that smallest values are white and largest black. The JPEG method of image compression uses an approach similar to the one shown here, though with basic forms that have continuous levels of gray, rather than just black and white.

what happens if one divides an image into a collection of nested squares, but imposes a lower limit on the size of these squares. And what one sees is that as the lower limit is increased, the amount of compression increases rapidly—though at the cost of a correspondingly rapid decrease in the quality of the image.

So can one do better at maintaining the quality of the image? Various schemes are used in practice, and almost all of them are based on the idea from traditional mathematics that by viewing data in terms of numbers it becomes possible to decompose the data into sums of fixed basic forms—some of which can be dropped in order to achieve compression.

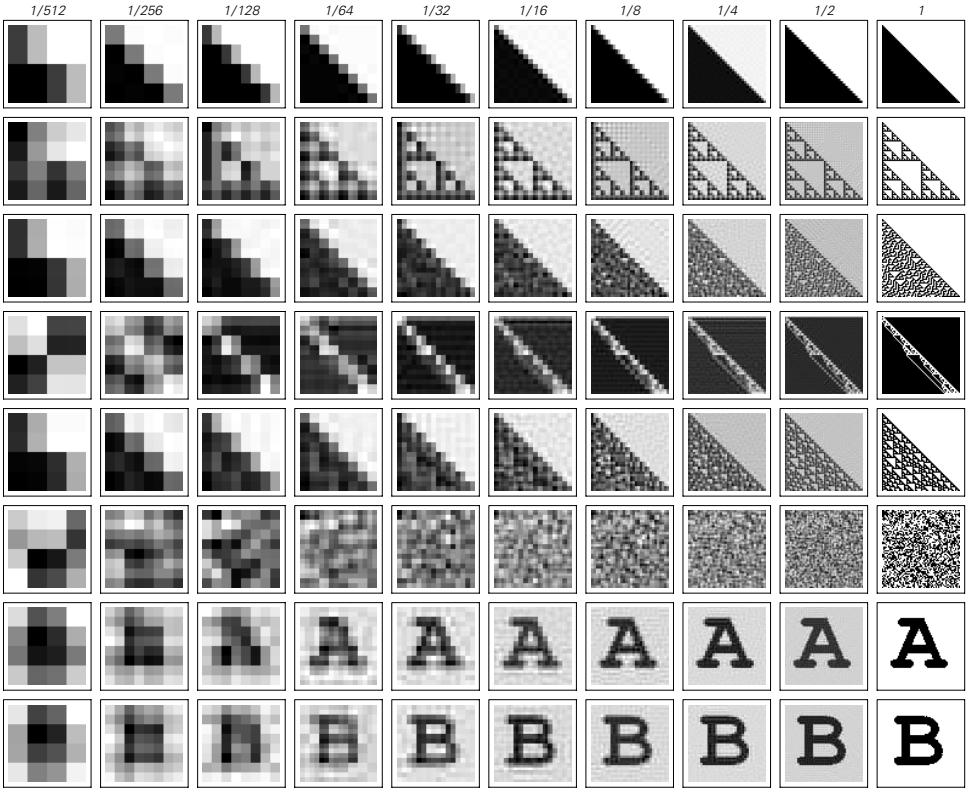
The pictures on the previous page show an example of how this works. On the top left is a set of basic forms which have the property that any two-dimensional image can be built up simply by adding together these forms with appropriate weights. On the top right these forms are then ranked roughly from coarsest to finest. And given this ranking, the arrays of pictures at the bottom show how two different images can be built up by progressively adding in more and more of the basic forms.

If all the basic forms are included, then the original image is faithfully reproduced. But if one drops some of the later forms—thereby reducing the number of weights that have to be specified—one gets only an approximation to the image. The facing page shows what happens to a variety of images when different fractions of the forms are kept.

Images that are sufficiently simple can already be recognized even when only a very small fraction of the forms are included—corresponding to a very high level of compression. But most other images typically require more forms to be included—and thus do not allow such high levels of compression.

Indeed the situation is very much what one would expect from the definition of complexity that I gave two sections ago. The relevant features of both simple and completely random images can readily be recognized even at quite high levels of compression. But images that one would normally consider complex tend to have features that cannot be recognized except at significantly lower levels of compression.

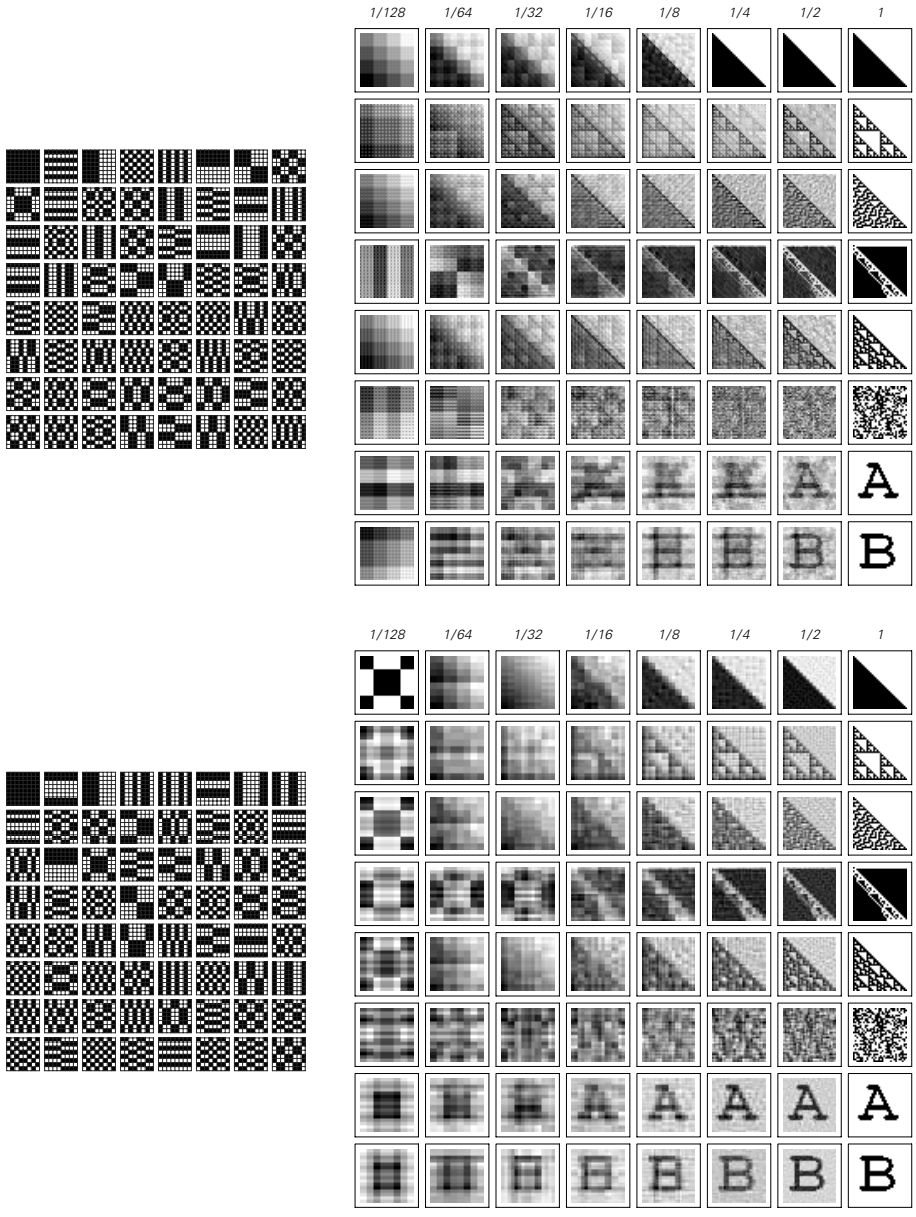
All the pictures on the facing page, however, were generated from the specific ordering of basic forms shown on the previous page. And




Examples of images obtained by keeping only certain fractions of the complete set of basic forms. In the case of both simple and completely random images, many features are recognizable even with fairly few basic forms—implying that a highly compressed representation can be given.

one might wonder whether perhaps some other ordering would make it easier to compress more complex images.

One simple approach is just to assemble a large collection of images typical of the ones that one wants to compress, and then to order the basic forms so that those which on average occur with larger weights in this collection appear first. The pictures on the next page show what happens if one does this first with images of cellular automata and then with images of letters. And indeed slightly higher levels of compression are achieved. But whatever ordering is used the fact seems to remain that images that we would normally consider complex still cannot systematically be compressed more than a small amount.




Results obtained by deducing optimal orderings of basic forms from collections of images of cellular automata (top) and letters (bottom). The orderings of basic forms are shown on the left, in each case starting with those whose weights are largest in absolute value when averaged over the collection of images. Note that the orderings are shown for $8 \times 8$ basic forms, while the actual images are $32 \times 32$. The orderings are deduced respectively from images of the 256 elementary cellular automata, and the 52 upper and lower case letters.

## Visual Perception

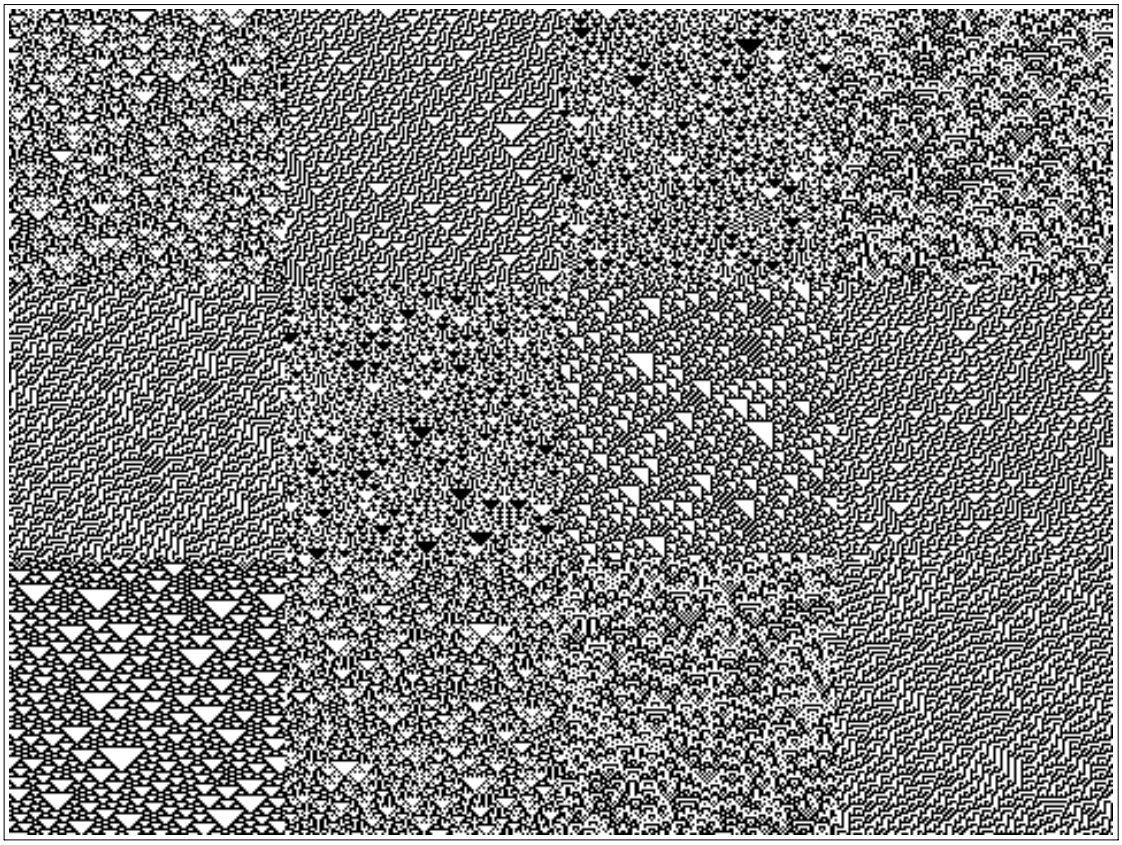
In modern times it has usually come to be considered quite unscientific to base very much just on how things look to our eyes. But the fact remains that despite all the various methods of mathematical and other analysis that have been developed, our visual system still represents one of the most powerful and reliable tools we have. And certainly in writing this book I have relied heavily on our ability to make all sorts of deductions on the basis of looking at visual representations.

So how does the human visual system actually work? And what are its limitations? There are many details yet to be resolved, but over the past couple of decades, it has begun to become fairly clear how at least the lowest levels of the system work. And it turns out—just as in so many other cases that we have seen in this book—that much of what goes on can be thought of in terms of remarkably simple programs.

In fact, across essentially every kind of human perception, the basic scheme that seems to be used over and over again is to have particular kinds of cells set up to respond to specific fixed features in the data, and then to ignore all other features.

Color perception provides a classic example. On the retina of our eye are three kinds of color-sensitive cells, with each kind responding essentially to the level of either red, green or blue. Light from an object typically involves a whole spectrum of wavelengths. But the fact that we have only three kinds of color-sensitive cells means that our eyes essentially sample only three features of this spectrum. And this is why, for example, we have the impression that mixtures of just three fixed colors can successfully reproduce all other colors.

So what about patterns and textures? Does our visual system also work by picking out specific features of these? Everyday experience suggests that indeed it does. For if we look, say, at the picture on the next page we do not immediately notice every detail. And instead what our visual system seems to do is just to pick out certain features which quickly make us see the picture as a collection of patches with definite textures.



Patches generated by a variety of one-dimensional cellular automaton rules. Each patch is set up to have a roughly equal number of black and white squares. But despite this, our visual system quickly notices that different patches have different textures. And presumably this is because the visual system is automatically identifying particular features in each patch. Everyone appears immediately to be able to see some patches when shown this picture. But after looking at the picture for a while, the boundaries between the patches seem to get somewhat clearer.

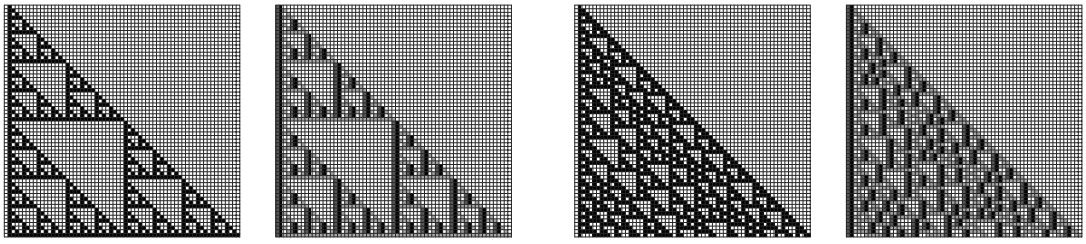
So how does this work? The basic answer seems to be that there are nerve cells in our eyes and brains which are set up to respond to particular local patterns in the image formed on the retina of our eye.

The way this comes about appears to be surprisingly direct. Behind the 100 million or so light-sensitive cells on our retina are a sequence of layers of nerve cells, first in the eye and then in the brain. The connections between these cells are set up so that a given cell in the visual cortex will typically receive inputs only from cells in a fairly small area on our retina. Some of these inputs will be positive if the

image in a certain part of the area is, say, colored white, while others will be positive if it is colored black. And the cell in the visual cortex will then respond only if enough of its inputs are positive, corresponding to a specific pattern being present in the image.

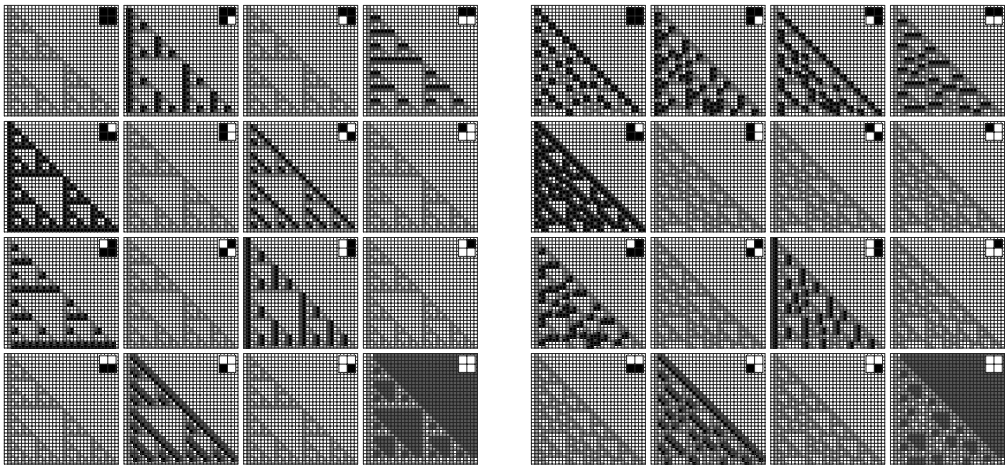
In practice many details of this setup are quite complicated. But as a simple idealization, one can consider an array of squares on the retina, each colored either black or white. And one can then assume that in the visual cortex there is a corresponding array of cells, with each cell receiving input from, say, a $2 \times 2$ block of squares, and following the rule that it responds whenever the colors of these squares form some particular pattern.

The pictures below show a simple example. In each case the first picture shows the image on the retina, while the second picture shows which cells respond to it. And with the specific choice of rule used here, what effectively happens is that the vertical black edges in the original image get picked out.



Responses to two sample images of cells sensitive to the $2 \times 2$ template shown on the left. The cells that respond are indicated by darker squares in the second picture in each pair. Such responses occur whenever the $2 \times 2$ template on the left appears, corresponding to the presence of a vertical black edge. The extraction of features by this kind of simple template matching appears to be a key element in human visual perception—as well as being common in technological image processing. The sample images used here are ones generated by the evolution of elementary one-dimensional cellular automata with rules 60 and 124 respectively.

Neurophysiological experiments suggest that cells in the visual cortex respond to a variety of specific kinds of patterns. And as a simple idealization, the pictures on the next page show what happens with cells that respond to each of the 16 possible $2 \times 2$ arrangements of black and white squares. In each case, one can think of the results as corresponding to picking out some specific local feature in the original image.

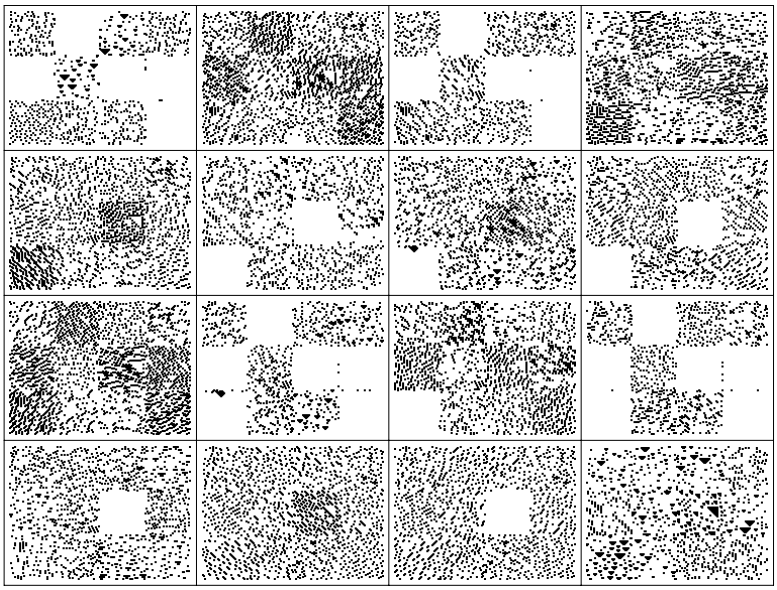


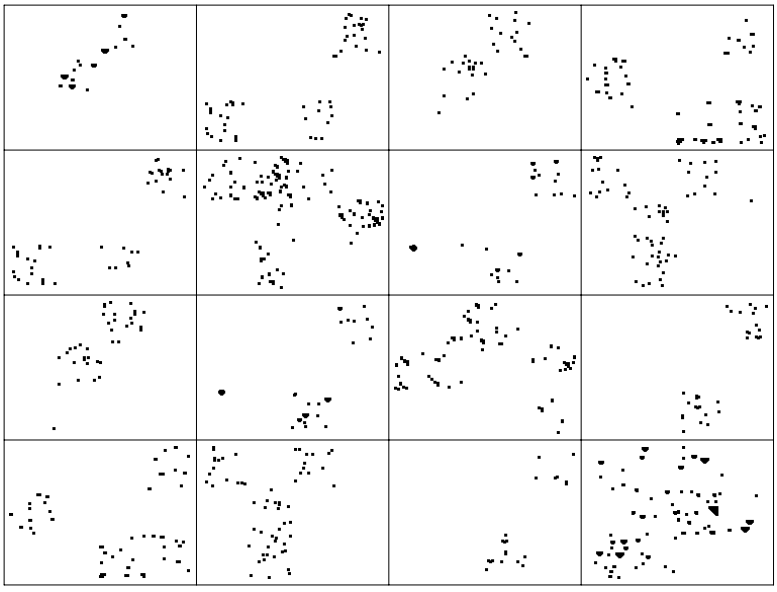
Responses to the sample images from the previous page by types of cells sensitive to each of the local arrangements of black and white squares shown. In each case, one can think of the resulting patterns as being filtered versions of the original images in which only parts that exhibit particular features are kept. The patterns can also be viewed as outputs from a single step in the evolution of two-dimensional block cellular automata in which the rules specify that a block becomes dark if it has the arrangement of cells shown, and becomes light otherwise. The comparative sparsity of dark blocks is a consequence of the fact that at any given position a dark block can occur in only one of the 16 cases shown. The absence of any dark blocks in many of the cases shown can be viewed as a reflection of constraints introduced by the construction of the images from one-dimensional cellular automaton rules.

So is this very simple kind of process really what underlies our seemingly sophisticated perception of patterns and textures? I strongly suspect that to a large extent it is. An important detail, however, is that there are cells in the visual cortex which in effect receive input from larger regions on the retina. But as a simple idealization one can assume that such cells in the end just respond to repeated versions of the basic $2 \times 2$ patterns.

So with this setup, the pictures on the facing page show what happens with an image like the one from page 578. The results are somewhat remarkable. For even though the average density of black and white squares is exactly the same across the whole image, what we see is that in different patches the features that end up being picked out have different densities. And it is this, I suspect, that makes us see different patches as having different textures.

For much as we distinguish colors by their densities of red, green and blue, so also it seems likely that we distinguish textures by their



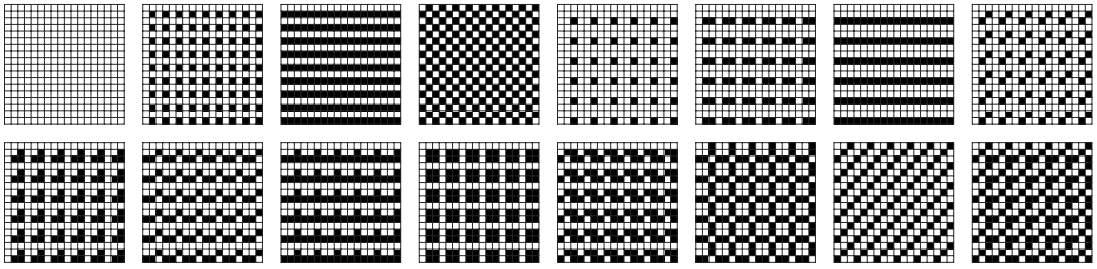


Responses to a smaller version of the image from page 578 by cells sensitive to all 16 possible 2 × 2 blocks, as well as their repetitive 3 × 3 extensions. Patches which appear to have different textures in the original image are seen to contain characteristically different densities of these various blocks. I strongly suspect that it is density differences such as these that allow our visual system to distinguish textures.

densities of certain local features. And the reason that this happens so quickly when we look at an image is no doubt that the procedure for picking out such features is a very simple one that can readily be carried out in parallel by large numbers of separate cells in our eyes and brains.

For patterns and textures, however, unlike for colors, we can always get beyond the immediate impression that our visual system provides. And so for example, by making a conscious effort, we can scan an image with our eyes, scrutinizing different parts in turn and comparing whatever details we want.

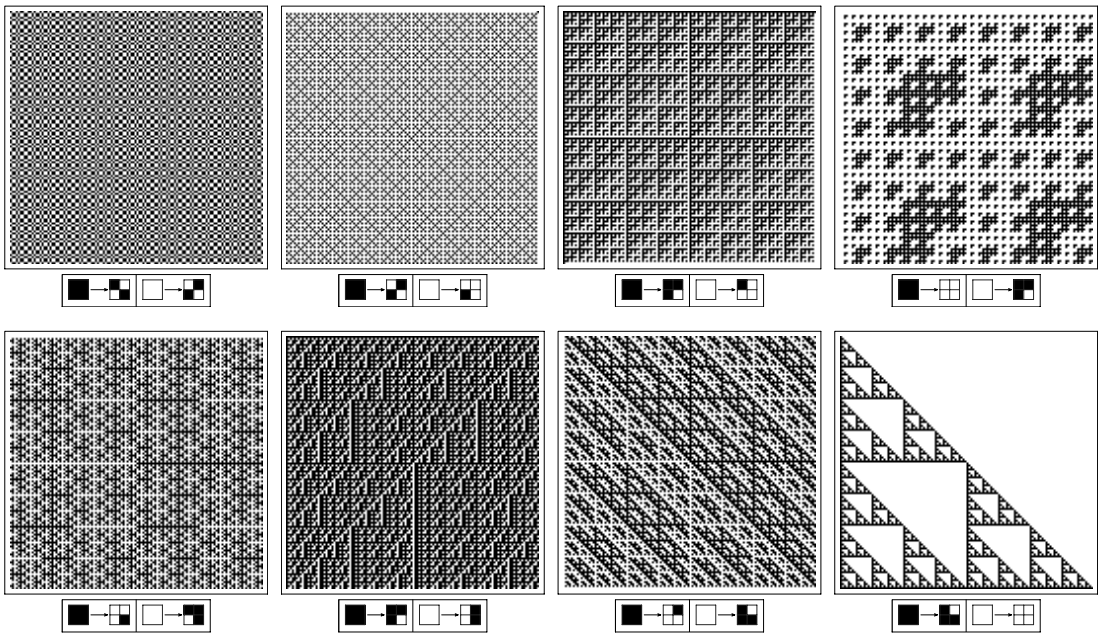
But what kinds of things can we expect to tell in this way? As the pictures below suggest, it is usually quite easy to see if an image is purely repetitive—even in cases where the block that repeats is fairly large.



Examples of all the distinct repetitive patterns that can be formed from arrays of 2 × 2 and 3 × 3 blocks. In every single case the presence of pure repetition is easy to recognize by eye. Note that in a pattern generated by repeating one particular block, there will normally be other blocks that occur with other alignments. Page 215 shows patterns obtained in systems based on constraints in which one effectively requires that only certain blocks or sets of blocks occur.

But with nesting the story is quite different. All eight pictures on the facing page were generated from the two-dimensional substitution systems shown, and thus correspond to purely nested patterns. But except for the last picture on each row—which happen to be dominated by large areas of essentially uniform color—it is remarkably difficult for us to tell that the patterns are nested. And this can be viewed as a clear example of a limitation in our powers of visual perception.

As we found two sections ago, many standard methods of data compression have the same limitation. But at the end of that section I showed that the fairly simple procedure of two-dimensional pointer



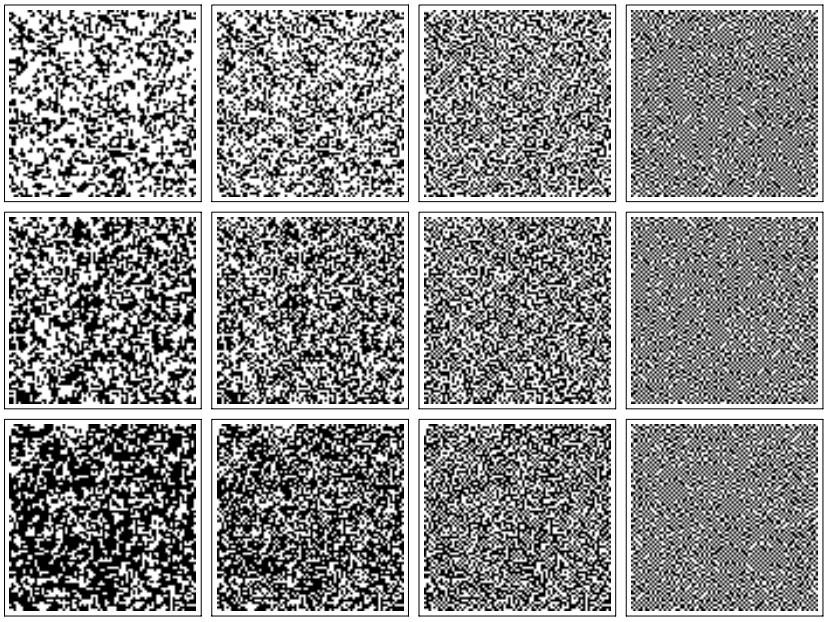
Examples of nested patterns created by following the two-dimensional substitution rules shown. Except for the last examples on each row, it is remarkably difficult to recognize the nested structure in these patterns by eye, even with quite careful scrutiny. The two-dimensional pointer-based encoding scheme from page 571 does however manage to recognize the structure in all cases.

encoding will succeed in recognizing nesting. So it is not that nesting is somehow fundamentally difficult to recognize; it is just that the particular processes that happen to occur in human visual perception do not in general manage to do it.

So what about randomness? The pictures on the next page show a few examples of images with various degrees of randomness. And just by looking at these images it is remarkably difficult to tell which of them is in fact the most random.

The basic problem is that our visual system makes us notice local features—such as clumps of black squares—even if their density is consistent with what it should be in a completely random array. And as a result, much as with constellations of stars, we tend to identify what seem to be regularities even in completely random patterns.

In principle it could be that there would be images in which our visual system would notice essentially no local features. And indeed in



Examples of images that approximate perfect randomness. The second image on each row has squares chosen independently to be black with probabilities 0.4, 0.5 and 0.6 respectively. In the other images various features are added or removed. In the first image on each row, if any square is surrounded by four squares with identical colors, then the square is forced to have the same color. In the third image, any clump of squares with the same color is broken up by reversing the color of the center square. And in the fourth image, the same is done with lines of squares of the same color.

the last two images on each row above all clumps of squares of the same color, and then all lines of squares of the same color, have explicitly been removed. At first glance, these images do in some respects look more random. But insofar as our visual system contains elements that respond to each of the possible local arrangements of squares, it is inevitable that we will identify features of some kind or another in absolutely any image.

In practice there are presumably some types of local patterns to which our visual system responds more strongly than others. And knowing such a hierarchy, one should be able to produce images that in a sense seem as random as possible to us. But inevitably such images would reflect much more the details of our process of visual perception than they would anything about actual underlying randomness.

## Auditory Perception

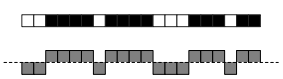
In the course of this book I have made extensive use of pictures. So why not also sounds? One issue—beyond the obvious fact that sounds cannot be included directly in a printed book—is that while one can study the details of a picture at whatever pace one wants, a sound is in a sense gone as soon as it has finished playing.

But everyday experience makes it quite clear that one can still learn a lot by listening to sounds. So what then are the features of sounds that our auditory system manages to pick out?

At a fundamental level all sounds consist of patterns of rapid vibrations. And the way that we hear sounds is by such vibrations being transmitted to the array of hair cells in our inner ear. The mechanics of the inner ear are set up so that each row of hair cells ends up being particularly sensitive to vibrations at some specific frequency. So what this means is that what we tend to perceive most about sounds are the frequencies they contain.

Musical notes usually have just one basic frequency, while voiced speech sounds have two or three. But what about sounds from systems in nature, or from systems of the kinds we have studied in this book?

There are a number of ways in which one can imagine such systems being used to generate sounds. One simple approach illustrated on the right is to consider a sequence of elements produced by the system, and then to take each element to correspond to a vibration for a brief time—say a thousandth of a second—in one of two directions.



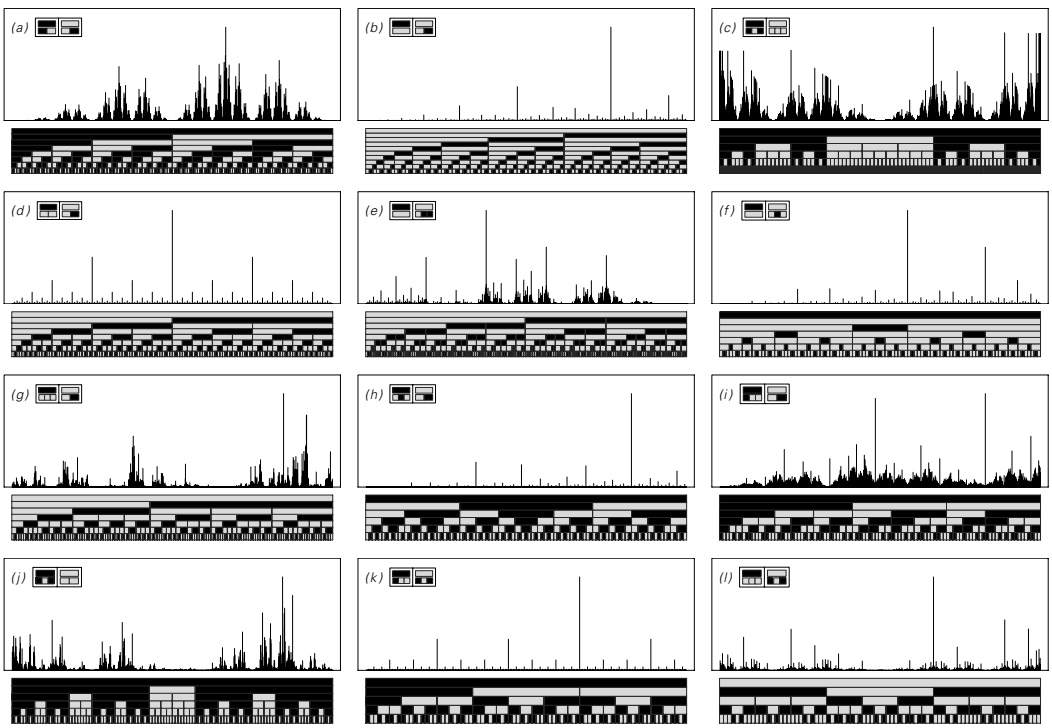
A sequence of discrete elements and a possible corresponding waveform for a sound.

So what are such sounds like? If the sequence of elements is repetitive then what one hears is in essence a pure tone at a specific frequency—much like a musical note. But if the sequence is random then what one hears is just an amorphous hiss.

So what happens between these extremes? If the properties of a sequence gradually change in a definite way over time then one can often hear this in the corresponding sound. But what about sequences that have more or less uniform properties? What kinds of regularities does our auditory system manage to detect in these?

The answer, it seems, is surprisingly simple: we readily recognize exact or approximate repetition at definite frequencies, and essentially nothing else. So if we listen to nested sequences, for example, we have no direct way to tell that they are nested, and indeed all we seem sensitive to are some rather simple features of the spectrum of frequencies that occur.

The pictures below show spectra obtained from nested sequences produced by various simple one-dimensional substitution systems. The diversity of these spectra is quite striking: some have simple nested forms dominated by a few isolated peaks at specific frequencies, while others have quite complex forms that cover large ranges of frequencies.

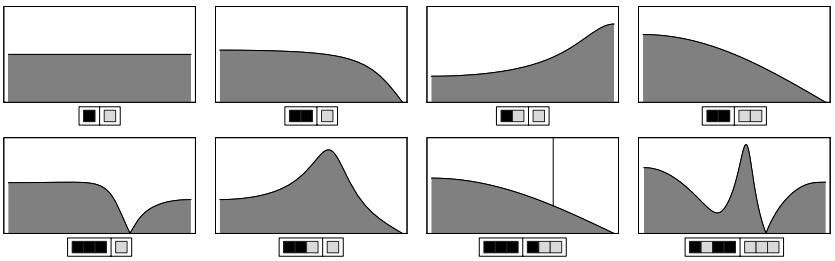



Frequency spectra of nested sequences generated by one-dimensional neighbor-independent substitution systems. The rules are the same as shown on pages 83 and 84. Note the presence of both isolated peaks and complicated background patterns. If a sequence corresponds to a pure tone and repeats every $n$ elements then its spectrum will consist of $n/2$ equally spaced peaks. Sequences whose spectra contain no dominant peaks typically sound like random noise, although sometimes explicit time variation can be heard, and indeed sequence (c) just sounds like a succession of idealized frog ribbets. Intensity or power spectra are obtained by squaring the quantities shown.

And given only the underlying rule for a substitution system, it turns out to be fairly difficult to tell even roughly what the spectrum will be like. But given the spectrum, one can immediately tell how we will perceive the sound. When the spectrum is dominated by just one large peak, we hear a definite tone. And when there are two large peaks we also typically hear definite tones. But as the number of peaks increases it rapidly becomes impossible to keep track of them, and we end up just hearing random noise—even in cases where the peaks happen to have frequencies that are in the ratios of common musical chords.

So the result is that our ears are not sensitive to most of the elaborate structure that we see in the spectra of many nested sequences. Indeed, it seems that as soon as the spectrum covers any broad range of frequencies all but very large peaks tend to be completely masked, just as in everyday life a sound needs to be loud if it is to be heard over background noise.

So what about other kinds of regularities in sequences? If a sequence is basically random but contains some short-range correlations then these will lead to smooth variations in the spectrum. And for example sequences that consist of random successions of specific blocks can yield any of the types of spectra shown below—and can sound variously like hisses, growls or gurgles.



Frequency spectra for long sequences obtained by concatenating blocks in random orders. Such spectra can be calculated by fairly standard methods from stochastic analysis. The first case shown corresponds to white noise. The second-to-last case always has a black element at every third position, so exhibits a peak at the corresponding repetition frequency.

To get a spectrum with a more elaborate structure requires long-range correlations—as exist in nested sequences. But so far as I can

tell, the only kinds of correlations that are ultimately important to our auditory system are those that lead to some form of repetition.

So in the end, any features of the behavior of a system that go beyond pure repetition will tend to seem to our ears essentially random.

## Statistical Analysis

When it comes to studying large volumes of data the method almost exclusively used in present-day science is statistical analysis. So what kinds of processes does such analysis involve? What is typically done in practice is to compute from raw data various fairly simple quantities whose values can then be used to assess models which could provide summaries of the data.

Most kinds of statistical analysis are fundamentally based on the assumption that such models must be probabilistic, in the sense that they give only probabilities for behavior, and do not specifically say what the behavior will be. In different situations the reasons for using such probabilistic models have been somewhat different, but before the discoveries in this book one of the key points was that it seemed inconceivable that there could be deterministic models that would reproduce the kinds of complexity and apparent randomness that were so often seen in practice.

If one has a deterministic model then it is at least in principle quite straightforward to find out whether the model is correct: for all one has to do is to compare whatever specific behavior the model predicts with behavior that one observes. But if one has a probabilistic model then it is a much more difficult matter to assess its validity—and indeed much of the technical development of the field of statistics, as well as many of its well-publicized problems, can be traced to this issue.

As one simple example, consider a model in which all possible sequences of black and white squares are supposed to occur with equal probability. By effectively enumerating all such sequences, it is easy to see that such a model predicts that in any particular sequence the fraction of black squares is most likely to be $1/2$.

But what if a sequence one actually observes has 9 black squares out of 10? Even though this is not the most likely thing to see, one certainly cannot conclude from seeing it that the model is wrong. For the model does not say that such sequences are impossible—it merely says that they should occur only about 1% of the time.

And indeed there is no meaningful way without more information to deduce any kind of absolute probability for the model to be correct. So in practice what almost universally ends up being done is to consider not just an individual model, but rather a whole class of models, and then to try to identify which model from this class is the best one—as measured, say, by the criterion that its likelihood of generating the observed data is as large as possible.

For sequences of black and white squares a simple class of models to consider are those in which each square is taken to be black with some fixed independent probability $p$. Given a set of raw data the procedure for finding which model in this class is best—according, say, to the criterion of maximum likelihood—is extremely straightforward: all one does is to compute what fraction of squares in the data are black, and this value then immediately gives the value of $p$ for the best model.

So what about more complicated models? Instead of taking each square to have a color that is chosen completely independently, one can for example take blocks of squares of some given length to have their colors chosen together. And in this case the best model is again straightforward to find: it simply takes the probabilities for different blocks to be equal to the frequencies with which these blocks occur in the data.

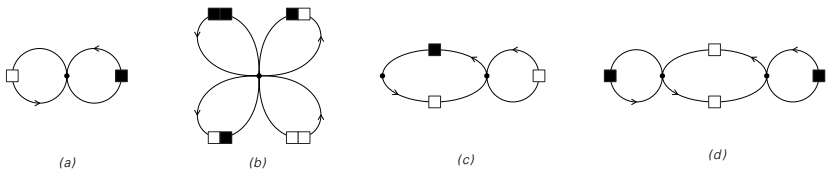
If one does not decide in advance how long the blocks are going to be, however, then things can become more complicated. For in such a case one can always just make up an extreme model in which only one very long block is allowed, with this block being precisely the sequence that is observed in the data.

Needless to say, such a model would for most purposes not be considered particularly useful—and certainly it does not succeed in providing any kind of short summary of the data. But to exclude models like this in a systematic way requires going beyond criteria such as

maximum likelihood, and somehow explicitly taking into account the complexity of the model itself.

For specific types of models it is possible to come up with various criteria based for example on the number of separate numerical parameters that the models contain. But in general the problem of working out what model is most appropriate for any given set of data is an extremely difficult one. Indeed, as discussed at the beginning of Chapter 8, it is in some sense the core issue in any kind of empirical approach to science.

But traditional statistical analysis is usually far from having to confront such issues. For typically it restricts itself to very specific classes of models—and usually ones which even by the standards of this book are extremely simple. For sequences of black and white squares, for example, models that work as above by just assigning probabilities to fixed blocks of squares are by far the most common. An alternative, typically viewed as quite advanced, is to assign probabilities to sequences by looking at the paths that correspond to these sequences in networks of the kind shown below.

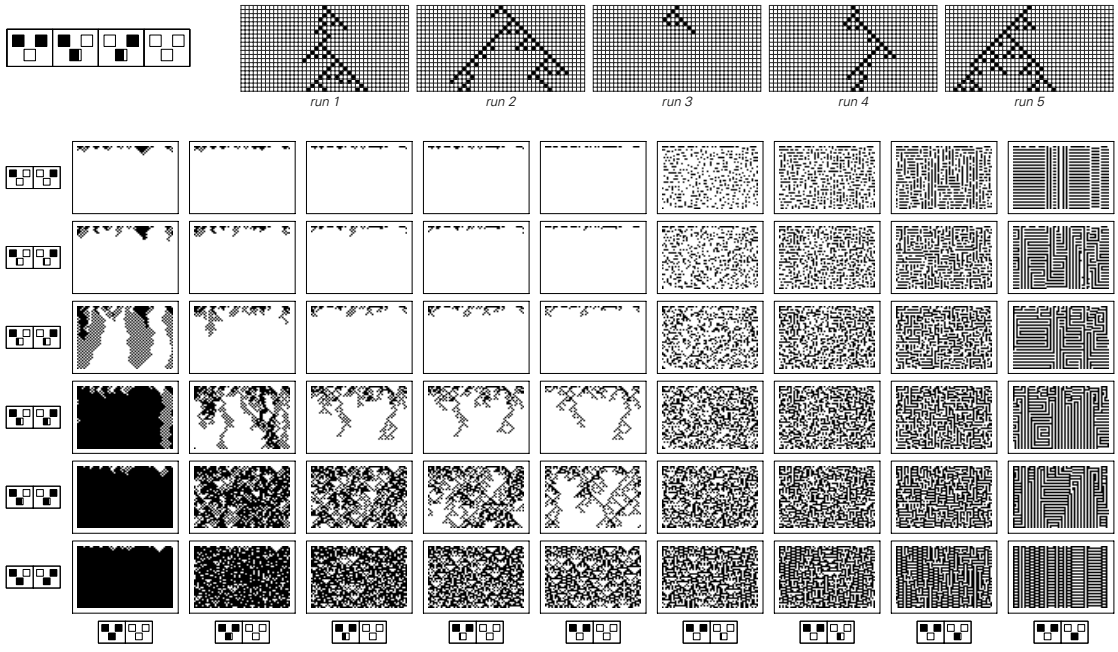



Networks defining probabilistic models. Each connection in each network has a certain probability associated with it, and the model takes sequences of black and white squares to be generated by tracing paths through the networks according to these probabilities. Cases (a) and (b) are so-called Markov models that in effect involve no memory and are equivalent to models discussed above. Cases (c) and (d) correspond to so-called hidden Markov models, with some short-term memory.

Networks (a) and (b) represent cases already discussed above. Network (a) specifies that the colors of successive squares should be chosen independently, while network (b) specifies that this should be done for successive pairs of squares. Network (c), however, specifies that different probabilities should be used depending on whether the path has reached the left or the right node in the network. But at least

so long as the structure of the network is kept the same, it is fairly easy even in this case to deduce from a given set of data what probabilities in the network provide the best model for the data—for essentially all one need do is to follow the path corresponding to the data, and see with what frequency each connection from each node ends up being used.

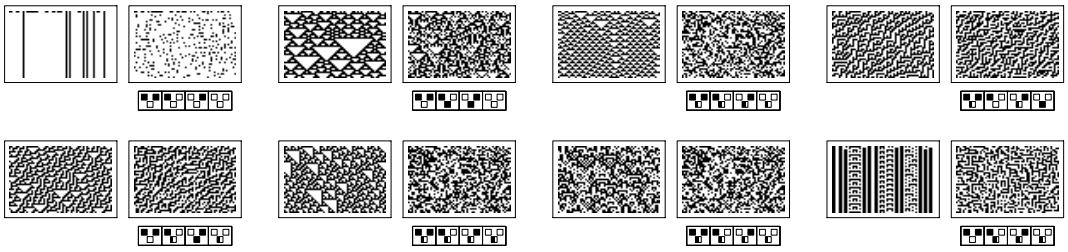
So what about two-dimensional data? From the discussion in Chapter 5 it follows that no straightforward analogs of the types of probabilistic models described above can be constructed in such a case. But as an alternative it turns out that one can use probabilistic versions of one-dimensional cellular automata, as in the pictures below.




Examples of probabilistic cellular automata, in which the rule specifies the probabilities for each color of cell to be generated given what the colors of its two neighbors were on the previous step. Because the rule is probabilistic a different detailed pattern of evolution will in general be obtained each time the cellular automaton is run—as in the top row of pictures above. Despite this, however, any particular probabilistic cellular automaton will typically exhibit some characteristic overall pattern of behavior, as illustrated in the array of pictures above. Note that it is fairly common for phase transitions to occur, in which continuous changes in underlying probabilities lead to discrete changes in typical behavior. Probabilistic cellular automata can be viewed as generalizations of so-called directed percolation models.

The rules for such cellular automata work by assigning to each possible neighborhood of cells a certain probability to generate a cell of each color. And for any particular form of neighborhood, it is once again quite straightforward to find the best model for any given set of data. For essentially all one need do is to work out with what frequency each color of cell appears below each possible neighborhood in the data.

But how good are the results one then gets? If one looks at quantities such as the overall density of black cells that were in effect used in finding the model in the first place then inevitably the results one gets seem quite good. But as soon as one looks at explicit pictures like the ones below, one immediately sees dramatic differences between the original data and what one gets from the model.



A comparison between data generated by ordinary cellular automata and the probabilistic cellular automata that are considered the best fit to it. While properties such as the density of black cells are typically set up to agree between the data and the model, the pictures make it clear that more detailed features do not.

In most cases, the typical behavior produced by the model looks considerably more random than the data. And indeed at some level this is hardly surprising: for by using a probabilistic model one is in a sense starting from an assumption of randomness.

The model can introduce certain regularities, but these almost never seem sufficient to force anything other than rather simple features of data to be correctly reproduced.

Needless to say, just as for most other forms of perception and analysis, it is typically not the goal of statistical analysis to find precise and complete representations of data. Rather, the purpose is usually just

to extract certain features that are relevant for drawing specific conclusions about the data.

And a fundamental example is to try to determine whether a given sequence can be considered perfectly random—or whether instead it contains obvious regularities of some kind.

From the point of view of statistical analysis, a sequence is perfectly random if it is somehow consistent with a model in which all possible sequences occur with equal probability.

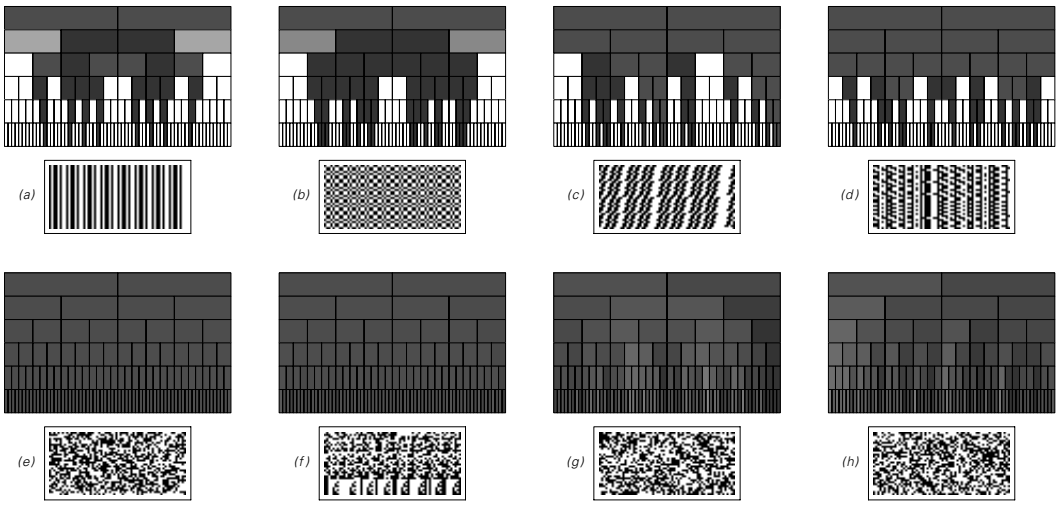
But how can one tell if this is so? What is typically done in practice is to take a sequence that is given and compute from it the values of various specific quantities, and then to compare these values with averages obtained by looking at all possible sequences.

Thus, for example, one might compute the fraction of squares in a given sequence that are black, and compare this to $1/2$. Or one might compute the frequency with which more than two consecutive black squares occur together, and compare this with the value $1/4$ obtained by averaging over all possible sequences.

And if one finds that a value computed from a particular sequence lies close to the average for all possible sequences then one can take this as evidence that the sequence is indeed random. But if one finds that the value lies far from the average then one can take this as evidence that the sequence is not random.

The pictures at the top of the next page show the results of computing the frequencies of different blocks in various sequences, and in each case each successive row shows results for all possible blocks of a given length. The gray levels on every row are set up so that the average of all possible sequences corresponds to the pattern of uniform gray shown below. So any deviation from such uniform gray potentially provides evidence for a deviation from randomness.

And what we see is that in the first three pictures, there are many obvious such deviations, while in the remaining pictures there are no obvious deviations. So from this it is fairly easy to conclude that the first three sequences are definitely not random, while the remaining sequences could still be random.




Statistics of block frequencies for various sequences. In each case the frequency of a particular block is represented by gray level, with results for blocks of successively greater lengths being shown on successive rows as indicated on the left. The original sequences are shown broken into lines and arranged in two dimensions. Sequences (b), (c) and (d) are generated by substitution systems with rules (b) ■ → ■□, □ → □■, (c) ■ → ■■□, □ → □ and (d) ■ → ■■■□□□, □ → ■□ respectively. (Note that these substitution systems are the simplest ones that yield equal frequencies of all blocks up to lengths 1, 2 and 3 respectively.) Sequence (e) is generated by a linear feedback shift register (essentially an additive cellular automaton) with tap positions *{2, 11}*. Sequence (f) is formed by concatenating base 2 digits of successive integers. Sequence (g) is the center column of the pattern generated by the rule 30 cellular automaton. Sequence (h) is the base 2 digits of $\pi$.

And indeed sequence (a) is certainly not random; in fact it is purely repetitive. And in general it is fairly easy to see that in any sequence that is purely repetitive there must beyond a certain length be many blocks whose frequencies are far from equal.

It turns out that the same is true for nested sequences. And in the picture above, sequences (b), (c) and (d) are all nested.

But what about the remaining sequences? Sequences (e) and (f) seem to yield frequencies that in every case correspond accurately to those obtained by averaging over all possible sequences. Sequences (g) and (h) yield results that are fairly similar, but exhibit some definite fluctuations.

So do these fluctuations represent evidence that sequences (g) and (h) are not in fact random? If one looks at the set of all possible sequences, one can fairly easily calculate the distribution of frequencies for any particular block. And from this distribution one can tell with

what probability a given deviation from the average should occur for a sequence that is genuinely chosen at random.

The result turns out to be quite consistent with what we see in pictures (g) and (h). But it is far from what we see in pictures (e) and (f). So even though individual block frequencies seem to suggest that sequences (d) and (e) are random, the lack of any spread in these frequencies provides evidence that in fact they are not.

So are sequences (g) and (h) in the end truly random? Just like other sequences discussed in this chapter they are in some sense not, since they can both be generated by simple underlying rules. But what the picture on the facing page demonstrates is that if one just does statistical analysis by computing frequencies of blocks one will see no evidence of any such underlying simplicity.

One might imagine that if one were to compute other quantities one could immediately find such evidence. But it turns out that many of the obvious quantities one might consider computing are in the end equivalent to various combinations of block frequencies. And perhaps as a result of this, it has sometimes been thought that if one could just compute frequencies of blocks of all lengths one would have a kind of universal test for randomness. But sequences like (e) and (f) on the facing page make it clear that this is not the case.

So what kinds of quantities can one in the end use in doing statistical analysis? The answer is that at least in principle one can use any quantity whatsoever, and in particular one can use quantities that arise from any of the processes of perception and analysis that I have discussed so far in this chapter. For in each case all one has to do is to compute the value of a quantity from a particular sequence of data, and then compare this value with what would be obtained by averaging over all possible sequences. In practice, however, the kinds of quantities actually used in statistical analysis of sequences tend to be rather limited. Indeed, beyond block frequencies, the only other ones that are common are those based on correlations, spectra, and occasionally run lengths—all of which we already discussed earlier in this chapter.

Nevertheless, one can in general imagine taking absolutely any process and using it as the basis for statistical analysis. For given some

specific process one can apply it to a piece of raw data, and then see how the results compare with those obtained from all possible sequences.

If the process is sufficiently simple then by using traditional mathematics one can sometimes work out fairly completely what will happen with all possible sequences. But in the vast majority of cases this cannot be done, and so in practice one has no choice but just to compare with results obtained by sampling some fairly limited collection of possible sequences.

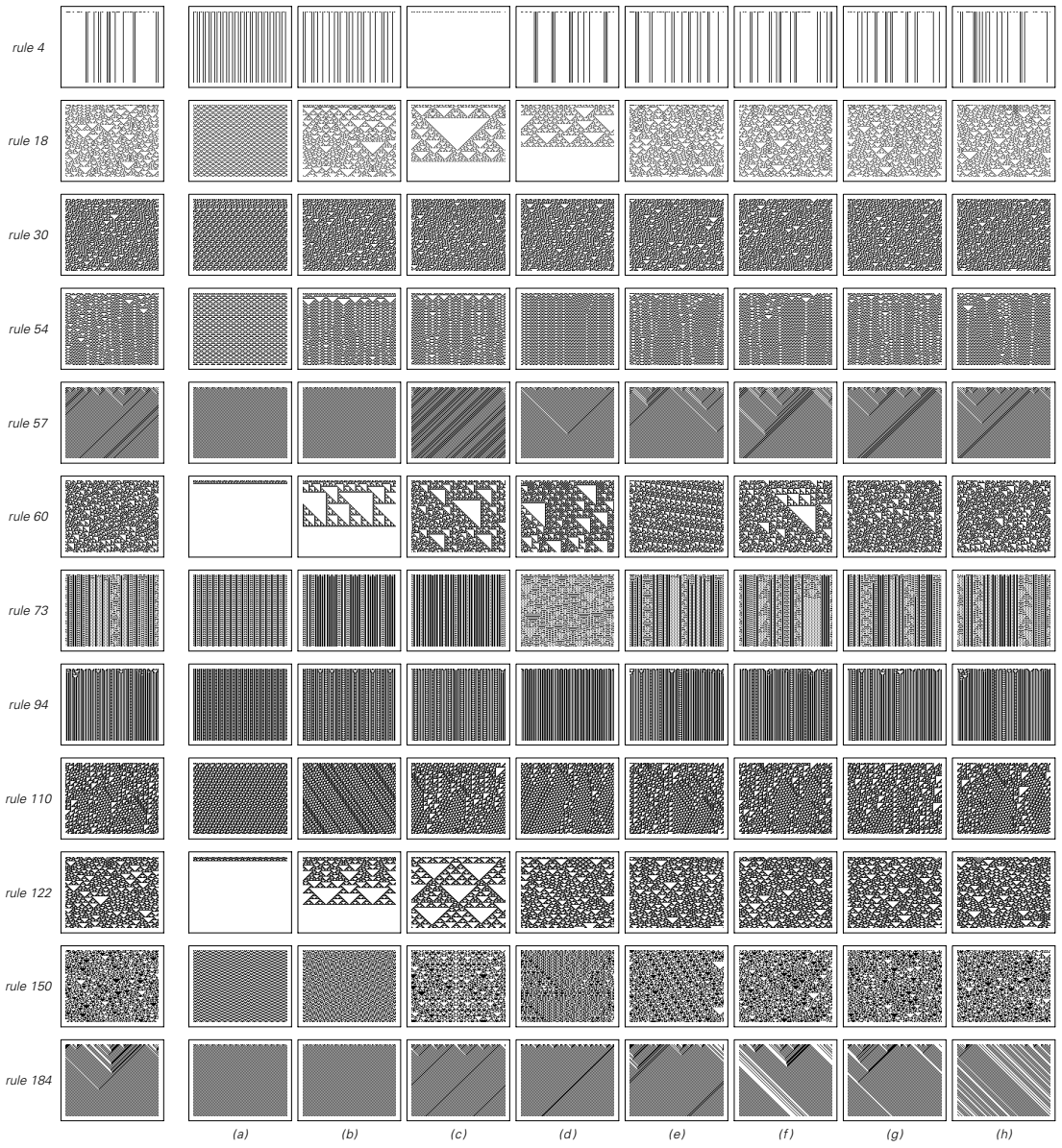
Under these circumstances therefore it becomes quite unrealistic to notice subtle deviations from average behavior. And indeed the only reliable strategy is usually just to look for cases in which there are huge differences between results for particular pieces of data and for typical sequences. For any such differences provide clear evidence that the data cannot in fact be considered random.

As an example of what can happen when simple processes are applied to data, the pictures on the facing page show the results of evolution according to various cellular automaton rules, with initial conditions given by the sequences from page 594. On each row the first picture illustrates the typical behavior of each cellular automaton. And the point is that if the sequences used as initial conditions for the other pictures are to be considered random then the behavior they yield should be similar.

But what we see is that in many cases the behavior actually obtained is dramatically different. And what this means is that in such cases statistical analysis based on simple cellular automata succeeds in recognizing that the sequences are not in fact random.

But what about sequences like (g) and (h)? With these sequences none of the simple cellular automaton rules shown here yield behavior that can readily be distinguished from what is typical. And indeed this is what I have found for all simple cellular automata that I have searched.

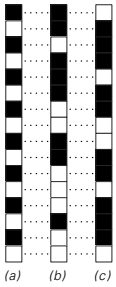
So from this we must conclude that—just as with all the other methods of perception and analysis discussed in this chapter—statistical analysis, even with some generalization, cannot readily recognize that sequences like (g) and (h) are anything but completely random—even though at an underlying level these sequences were generated by quite simple rules.




Examples of applying various rules for cellular automaton evolution to the sequences from page 594. The picture at the left-hand end of each row is chosen to show the typical behavior of each cellular automaton, given arbitrary initial conditions. Each cellular automaton rule in effect corresponds to a different statistical analysis procedure. Rule 4 picks out isolated black cells. Rule 60 essentially constructs a difference table for the sequence of elements. Rules 57 and 184 test for the overall density of black cells. (As indicated by page 136 the preponderance of white stripes with rule 184 in case (h) is a fluctuation.)

## Cryptography and Cryptanalysis

The purpose of cryptography is to hide the contents of messages by encrypting them so as to make them unrecognizable except by someone who has been given a special decryption key. The purpose of cryptanalysis is then to defeat this by finding ways to decrypt messages without being given the key.




Example of a scheme for encryption. From the original message (a) an encrypted message (c) is generated by reversing the color of each square for which the corresponding square in the encrypting sequence (b) is black. This scheme is the basis for essentially all practical stream ciphers.

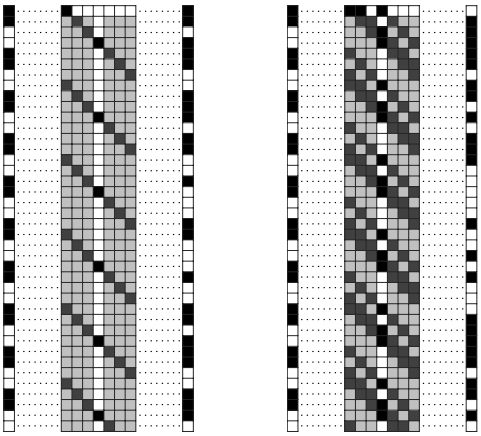
The picture on the left shows a standard method of encrypting messages represented by sequences of black and white squares. The basic idea is to have an encrypting sequence, shown as column (b) on the left, and from the original message (a) to get an encrypted version of the message (c) by reversing the color of every square for which the corresponding square in the encrypting sequence (b) is black.

So if one receives the encrypted message (c), how can one recover the original message (a)? If one knows the encrypting sequence (b) then it is straightforward. For all one need do is to repeat the process that was used for encryption, and reverse the color of every square in (c) for which the corresponding square in (b) is black.

But how can one arrange that only the intended recipient of the message knows the encrypting sequence (b)? In some situations it may be feasible to transmit the whole encrypting sequence in some secure way. But much more common is to be able to transmit only some short key in a secure way, and then to have to generate the encrypting sequence from this key.

So what kind of procedure might one use to get an encrypting sequence from a key? The picture at the top of the facing page shows an extremely simple approach that was widely used in practical cryptography until less than a century ago. The idea is just to form an encrypting sequence by repeatedly cycling through the elements in the key. And as the picture demonstrates, combining this with the original message leads to an encrypted message in which at least some of the structure in the original message is obscured.

But perhaps not surprisingly it is fairly easy to do cryptanalysis in such a case. For if one can find out what any sufficiently long segment in the encrypting sequence was, then this immediately gives the key,



A simple example of an encryption system in which the encrypting sequence is obtained by repetitively cycling through the elements of the key. Encryption with two different keys is shown. In each case the original message is on the left, the encrypted message is on the right, and the encrypting sequence corresponds to the highlighted column of cells. The system is essentially a Vigenère cipher of the kind widely used between the 1500s and the early 1900s.

and from the key the whole of the rest of the encrypting sequence can immediately be generated.

So what kind of analysis is needed to find a segment of the encrypting sequence? In an extreme but in practice common case one might happen to know what certain parts of the original message were—perhaps standardized greetings or some such—and by comparing the original and encrypted forms of these parts one can immediately deduce what the corresponding parts of the encrypting sequence must have been.

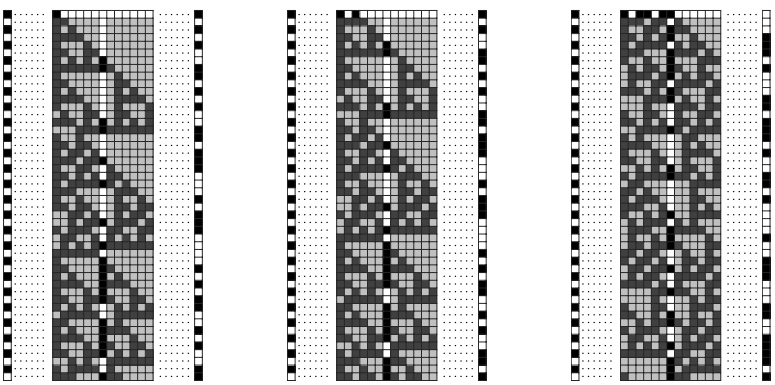
And even if all one knows is that the original message was in some definite language this is still typically good enough. For it means that there will be certain blocks—say corresponding to words like "the" in English—that occur much more often than others in the original message. And since such blocks must be encrypted in the same way whenever they occur at the same point in the repetition period of the encrypting sequence they will lead to occasional repeats in the encrypted message—with the spacing of such repeats always being some multiple of the repetition period. So this means that just by looking at the distribution of spacings between repeats one can expect to determine the repetition period of the encrypting sequence.

And once this is known, it is usually fairly straightforward to find the actual key. For one can pick out of the encrypted message all the squares that occur at a certain point in the repetition period of the

encrypting sequence, and which are therefore encrypted using a particular element of the key. Then one can ask whether such squares are more often black or more often white, and one can compare this with the result obtained by looking at the frequencies of letters in the language of the original message. If these two results are the same, then it suggests that the corresponding element in the key is white, and if they are different then it suggests that it is black. And once one has found a candidate key it is easy to check whether the key is correct by trying to use it to recover some reasonably long part of the original message. For unless one has the correct key, the chance that what one recovers will be meaningful in the language of the original message is absolutely negligible.

So what happens if one uses a more complicated rule for generating an encrypting sequence from a key? Methods like the ones above still turn out to allow features of the encrypting sequence to be found. And so to make cryptography work it must be the case that even if one knows certain features or parts of the encrypting sequence it is still difficult to deduce the original key or otherwise to generate the rest of the sequence.

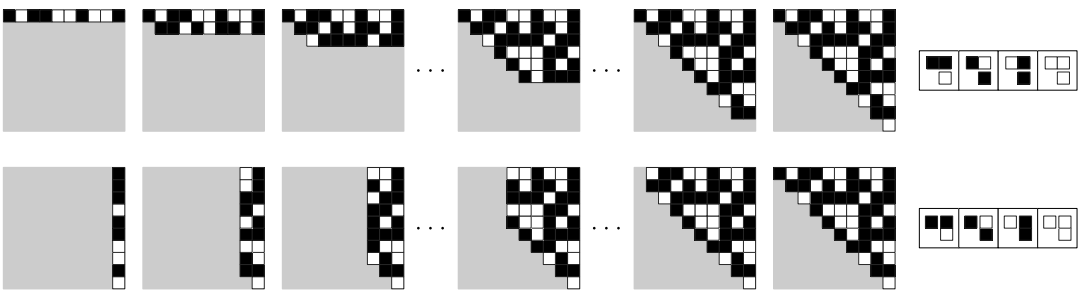
The picture below shows one way of generating encrypting sequences that was widely used in the early years of electronic cryptography, and is still sometimes used today. The basic idea is to look at the evolution of an additive cellular automaton in a register of limited width. The key then gives the initial condition for the cellular automaton, and the encrypting sequence is extracted, for example, by sampling a particular cell on successive steps.



Encryption using the rule 60 additive cellular automaton. This is essentially equivalent to a linear feedback shift register.

So given such an encrypting sequence, is there any easy way to do cryptanalysis and go backwards and work out the key?

It turns out that there is. For as the picture below demonstrates, in an additive cellular automaton like the one considered here the underlying rule is such that it allows one not only to deduce the form of a particular row from the row above it, but also to deduce the form of a particular column from the column to its right. And what this means is that if one has some segment of the encrypting sequence, corresponding to part of a column, then one can immediately use this to deduce the forms of a sequence of other columns, and thus to find the form of a row in the cellular automaton—and hence the original key.

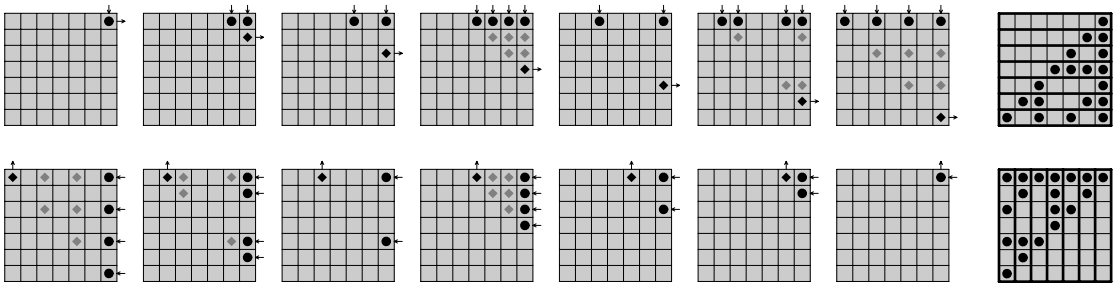


An example of the basis for cryptanalysis of an additive cellular automaton. The first set of pictures show the ordinary evolution of the rule 60 cellular automaton, in which each successive row is deduced from the one above. The second set of pictures show a kind of sideways evolution in which the rule is reinterpreted so as to allow a column of cells to be deduced from the column immediately to its right. Note that in both cases the colors of cells in the area on the lower right cannot be determined without knowing the colors of more initial cells than are shown.

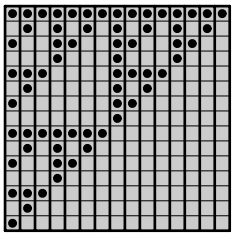
But what happens if the encrypting sequence does not include every single cell in a particular column? One cannot then immediately use the method described above. But it turns out that the additive nature of the underlying rule still makes comparatively straightforward cryptanalysis possible.

The picture on the next page shows how this works. Because of additivity it turns out that one can deduce whether or not some cell a certain number of steps down a given column is black just by seeing whether there are an odd or even number of black cells in certain specific positions in the row at the top. And one can then immediately

invert this to get a way to deduce the colors of cells on a given row from the colors of certain combinations of cells in a given column.



Another consequence of additivity: the correspondence between colors of cells on rows and columns in the rule 60 cellular automaton. In each case specifying the colors of the cells that are marked with dots immediately determines the colors of the cells that are marked with diamonds. The final diamond cell is black if an odd number of the dotted cells are black, and is white otherwise. The pictures on the right show which cells in the top row and which cells in the right-hand column determine the cells at successive positions in the right-hand column and in the top row respectively. These pictures can be thought of as matrices with 1's at the position of each black dot, and 0's elsewhere. Multiplying these matrices modulo 2 by vectors corresponding to a row of the cellular automaton gives a column, and vice versa. This means that the matrix on the second row of pictures is the inverse modulo 2 of the one on the first row.



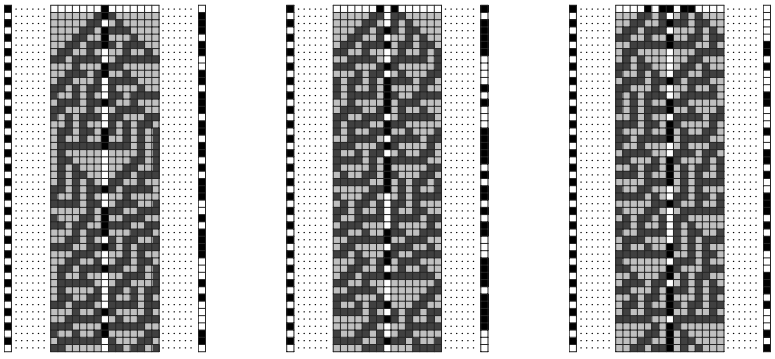
Which cells in a column are known will depend on how the encrypting sequence was formed. But with almost any scheme it will eventually be possible to determine the colors of cells at each of the positions across any register of limited width. So once again a fairly simple process is sufficient to allow the original key to be found.

So how then can one make a system that is not so vulnerable to cryptanalysis? One approach often used in practice is to form combinations of rules of the kind described above, and then to hope that the complexity of such rules will somehow have the effect of making cryptanalysis difficult.

But as we have seen many times in this book, more complicated rules do not necessarily produce behavior that is fundamentally any more complicated. And instead what we have discovered is that even among extremely simple rules there are ones which seem to yield behavior that is in a sense as complicated as anything.

So can such rules be used for cryptography? I strongly suspect that they can, and that in fact they allow one to construct systems that are at least as secure to cryptanalysis as any that are known.

The picture below shows a simple example based on the rule 30 cellular automaton that I have discussed several times before in this book. The idea is to generate an encrypting sequence by sampling the evolution of the cellular automaton, starting from initial conditions that are defined by a key.



Encryption using a column of rule 30 as the encrypting sequence. I first suggested this method in 1985.

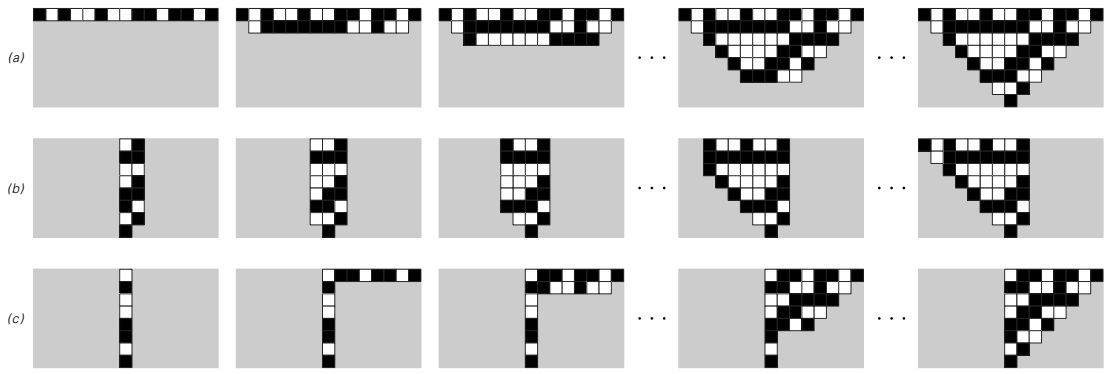
In the case of the additive cellular automaton shown on the previous page its nested structure makes it possible to recognize regularities using many of the methods of perception and analysis discussed in this chapter. But with rule 30 most sequences that are generated—even from simple initial conditions—appear completely random with respect to all of the methods of perception and analysis discussed so far.

So what about cryptanalysis? Does this also fail to find regularities, or does it provide some special way—at least within the context of a setup like the one shown above—to recognize whatever regularities are necessary for one to be able to deduce the initial condition and thus determine the key?

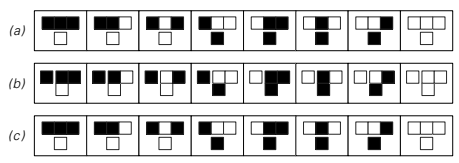
There is one approach that will always in principle work: one can just enumerate every possible initial condition, and then see which of them yields the sequence one wants. But as the width of the cellular automaton increases, the total number of possible initial conditions

rapidly becomes astronomical, and to test all of them becomes completely infeasible.

So are there other approaches that can be used? It turns out that as illustrated in the picture below rule 30 has a property somewhat like the additive cellular automaton discussed two pages ago: in addition to allowing one row to be deduced from the row above, it allows columns to be deduced from columns to their right. But unlike for the additive cellular automaton, it takes not just one column but instead two adjacent columns to make this possible.



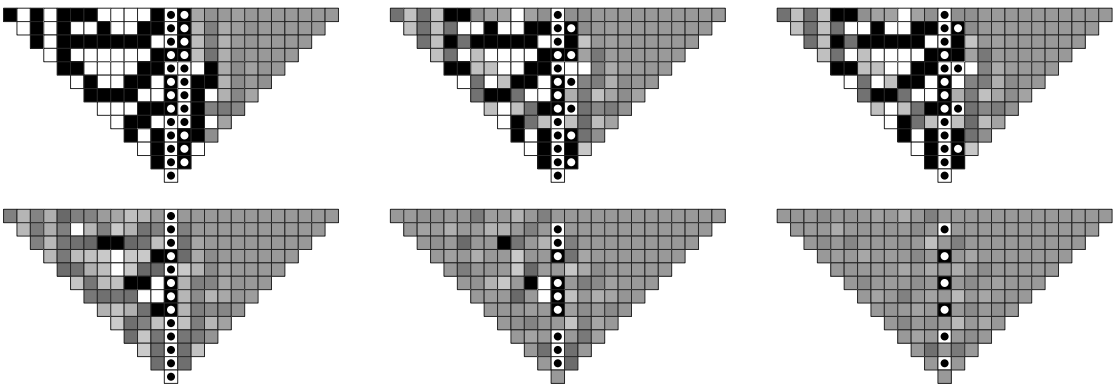

Sideways evolution in rule 30. (a) shows ordinary evolution from one row to the next. (b) shows evolution to the left starting from a pair of adjacent columns. (c) shows how a second column can be filled in from a row of cells to the right. The possibility of (b) is a consequence of one-sided additivity in rule 30; it leads to some level of cryptanalysis if the encrypting sequence consists of a complete column of cells.




So if the encrypting sequence corresponds to a single column, how can one find an adjacent column? The last row of pictures above show a way to do this. One picks some sequence of cells for the right half of the top row, then evolves down the page. And somewhat surprisingly, it turns out that given the cells in one column, there are fairly few possibilities for what the neighboring column can be. So by sampling a limited number of sequences on the top row, one can often find a second column that then allows columns to the left to be determined, and thus for a candidate key to be found.

But it is rather easy to foil this particular approach to cryptanalysis: all one need do is not sample every single cell in a given column in forming the encrypting sequence. For without every cell there does not appear to be enough information for any kind of local rule to be able to deduce one column from others.

The picture below shows evidence for this. The cells marked by dots have colors that are taken as given, and then the colors of other cells are filled in according to the average that is obtained by starting from all possible initial conditions.



Patterns generated by rule 30 after averaging over all possible initial conditions that reproduce the arrangements of colors in the cells indicated by dots. If a cell is completely black or completely white then this means that its color is uniquely determined by the constraints given. If the cell is shown as gray then this means that it has some probability of being black and some probability of being white. Note that when two complete adjacent columns are specified all the cells on the left-hand side are determined. But when fewer cells are specified, the number of cells that are determined decreases rapidly, indicating that cryptanalysis is likely to become difficult.

With two complete columns given, all cells to the left are determined to be either black or white. And with one complete column given, significant patches of cells still have determined colors. But if only every other cell in a column is given, almost nothing definite follows about the colors of other cells.

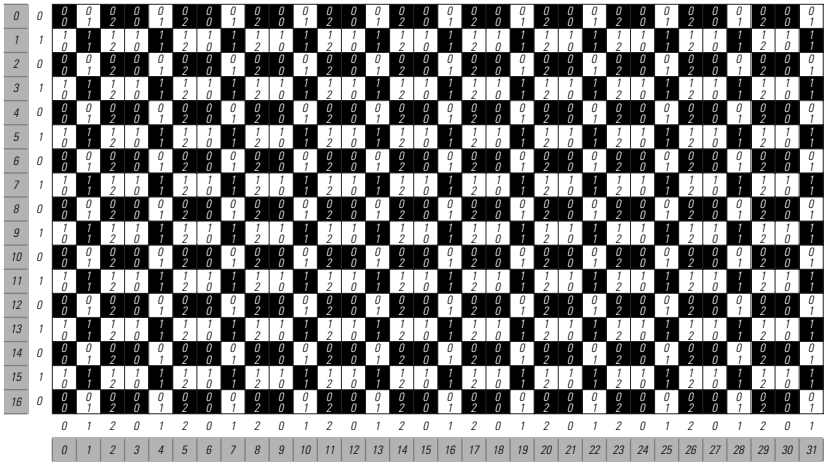
So what about the approach on page 602? Could this not be used here? It turns out that the approach relies crucially on the additivity of the underlying rules. And since rule 30 is not additive, it simply does not work. What happens is that the function that determines the color of a particular cell from the colors of cells in a nearby column rapidly becomes extremely

complicated—so that the approach probably ends up essentially being no better than just enumerating possible initial conditions.

The conclusion therefore is that at least with standard methods of cryptanalysis—as well as a few others—there appears to be no easy way to deduce the key for rule 30 from any suitably chosen encrypting sequence. But how can one be sure that there really is absolutely no easy way to do this? In Chapter 12 I will discuss some fundamental approaches to such a question. But as a practical matter one can say that not only have direct attempts to find easy ways to deduce the key in rule 30 failed, but also—despite some considerable effort—little progress has been made in solving any of various problems that turn out to be equivalent to this one.

## Traditional Mathematics and Mathematical Formulas

Traditional mathematics has for a long time been the primary method of analysis used throughout the theoretical sciences. Its goal can usually be thought of as trying to find a mathematical formula that summarizes the behavior of a system. So in a simple case if one has an array of black and white squares, what one would typically look for is a formula that takes the numbers which specify the position of a particular square and from these tells one whether the square is black or white.



With a pattern that is purely repetitive, the formula is always straightforward, as the picture at the bottom of the facing page illustrates. For all one ever need do is to work out the remainder from dividing the position of a particular square by the size of the basic repeating block, and this then immediately tells one how to look up the color one wants.

So what about nested patterns? It turns out that in most of traditional mathematics such patterns are already viewed as quite advanced. But with the right approach, it is in the end still fairly straightforward to find formulas for them.

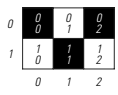
The crucial idea—much as in Chapter 4—is to think about numbers not in terms of their size but instead in terms of their digit sequences. And with this idea the picture on the next page shows an example of how what is in effect a formula can be constructed for a nested pattern.

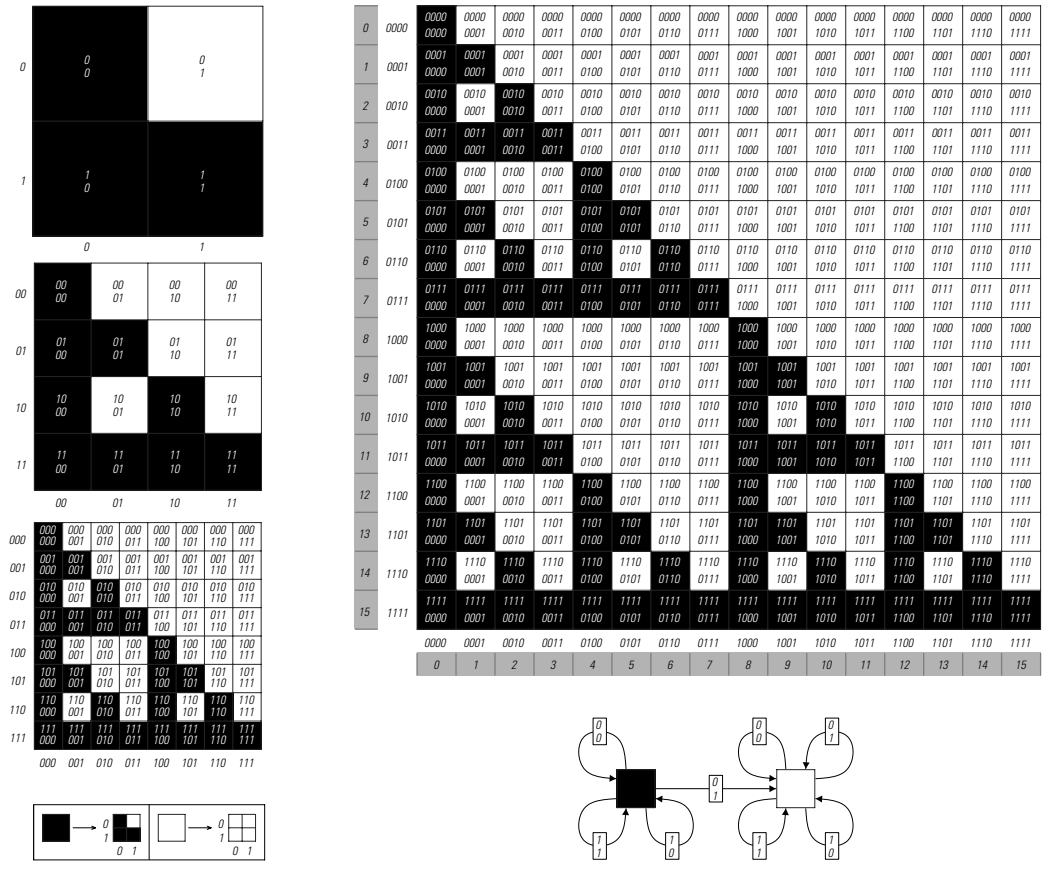
What one does is to look at the digit sequences for the numbers that give the vertical and horizontal positions of a certain square. And then in the specific case shown one compares corresponding digits in these two sequences, and if these digits are ever respectively 0 and 1, then the square is white; otherwise it is black.

So why does this procedure work?

As we have discussed several times in this book, any nested pattern must—almost by definition—be able to be reproduced by a neighbor-independent substitution system. And in the case shown on the next page the rules for this system are such that they replace each square at each step by a $2 \times 2$ block of new squares. So as the picture illustrates this means that new squares always have positions that involve numbers containing one extra digit. With the particular rules shown, the new squares always have the same color as the old one, except in one specific case: when a black square is replaced, the new square that appears in the upper right is always white. But this square

◀ An example of how the color of any square in a repetitive pattern can be found from its coordinates by a simple mathematical procedure. The procedure takes the *x* and *y* coordinates of the square, and computes their remainders after division by 3 and 2 respectively. Using these remainders—which are shown inside each square—the color of a particular square can be determined by a simple lookup in the repeating block shown on the left. The whole procedure can be represented using a mathematical formula that involves either functions like *Mod* or more traditional functions like *Sin*.



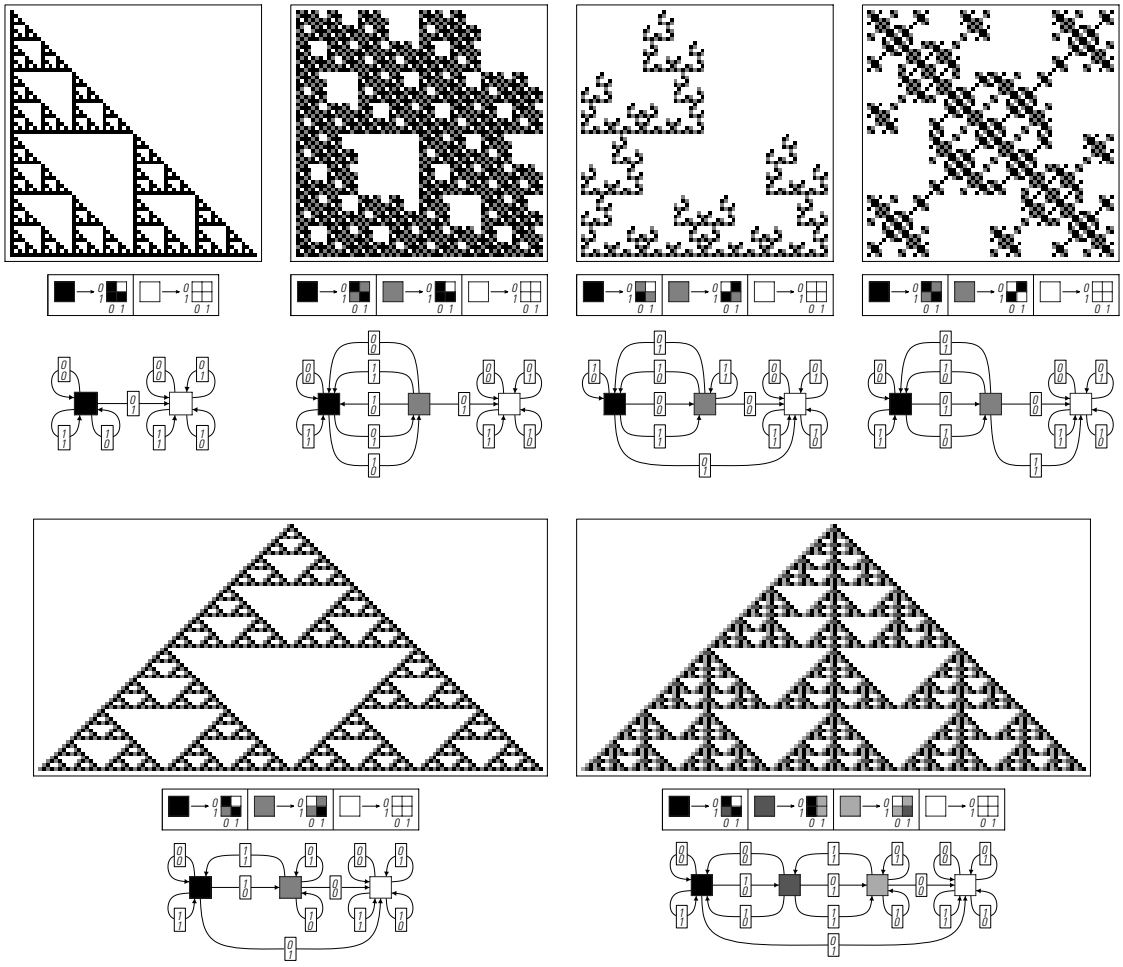


An example of how the color of any square in a nested pattern can be found from its coordinates by a fairly simple mathematical procedure. The procedure works by looking at the base 2 digit sequences of the coordinates. If any digit in the $y$ coordinate of a particular square is 0 when the corresponding digit in the $x$ coordinate is 1 then the square is white; otherwise it is black. The finite automaton at the bottom right gives a representation of this rule. Starting from the black square, one follows the sequence of connections that corresponds to the successive digits that one encounters in the $y$ and $x$ coordinates. Whatever square one lands up at in the finite automaton then gives the color one wants. Why this procedure works is illustrated by the pictures on the left. The nested pattern can be built up by a 2D substitution system with the rules shown. At each step in the evolution of this substitution system one gets a finer grid of squares, each specified in effect by one more digit in their coordinates.

has the property that its vertical position ends with a 0, and its horizontal position ends with a 1. So if the numbers that correspond to the position of a particular square contain this combination of digits at any point, it follows that the square must be white.

So what about other nested patterns? It turns out that using an extension of the argument above it is always possible to take the rules

for the substitution system that generates a particular nested pattern, and from these construct a procedure for finding the color of a square in the pattern given its position. The pictures below show several examples, and in all cases the procedures are fairly straightforward.



Procedures for determining the color of a square at a given position in various nested patterns. In each case the whole pattern can be generated by repeatedly applying the substitution system rule shown. The color of any particular square can also be found by feeding the digit sequences of its $y$ and $x$ coordinates to the finite automaton shown. The first example shown corresponds to cellular automaton rule 60; the last two examples correspond respectively to rules 90 and 150. In the top row of examples, the initial condition for the substitution system is a single black square, and the start state for the finite automaton is also its black state. In the second row of examples, the initial condition consists of a light gray square next to a black square. In these cases, the colors of squares to the left of the center can be found by starting from the light gray state in the finite automaton; the colors of squares to the right can be found by starting from the black state.

But while these procedures could easily be implemented as programs, they are in a sense not based on what are traditionally thought of as ordinary mathematical functions. So is it in fact possible to get formulas for the colors of squares that involve only such functions?

In the one specific case shown at the top of the facing page it turns out to be fairly easy. For it so happens that this particular pattern—which is equivalent to the patterns at the beginning of each row on the previous page—can be obtained just by adding together pairs of numbers in the format of Pascal's triangle and then putting a black square whenever there is an entry that is an odd number.

And as the table below illustrates, the entries in Pascal's triangle are simply the binomial coefficients that appear when one expands out the powers of $1 + x$. So to determine whether a particular square in the pattern is black or white, all one need do is to compute the corresponding binomial coefficient, and see whether or not it is an odd number. And this means that if black is represented by 1 and white by 0, one can then give an explicit formula for the color of the square at position $x$ on row $y$: it is simply (1 – (–1) ^ Binomial[$y$, $x$])/2.

| | |
|---|---|
| $1$ | $1$ |
| $1+x$ | $1+x$ |
| $(1+x)^2$ | $1+2x+x^2$ |
| $(1+x)^3$ | $1+3x+3x^2+x^3$ |
| $(1+x)^4$ | $1+4x+6x^2+4x^3+x^4$ |
| $(1+x)^5$ | $1+5x+10x^2+10x^3+5x^4+x^5$ |

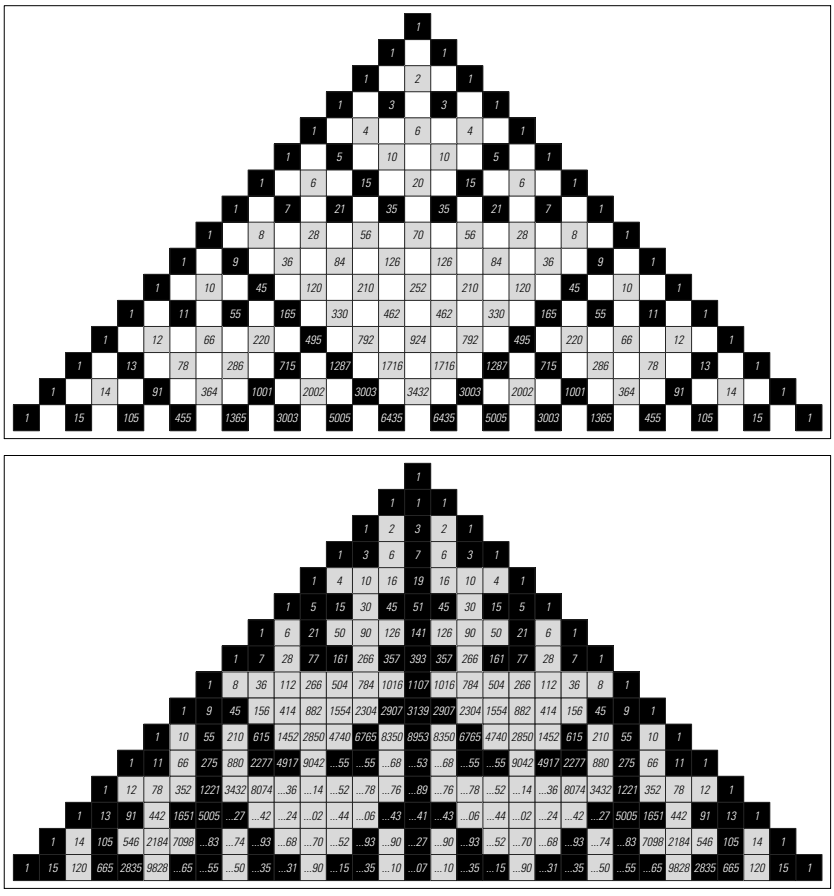
*Binomial[t, n]*

| | |
|---|---|
| $1$ | $1$ |
| $1+x+x^2$ | $1+x+x^2$ |
| $(1+x+x^2)^2$ | $1+2x+3x^2+2x^3+x^4$ |
| $(1+x+x^2)^3$ | $1+3x+6x^2+7x^3+6x^4+3x^5+x^6$ |
| $(1+x+x^2)^4$ | $1+4x+10x^2+16x^3+19x^4+16x^5+10x^6+4x^7+x^8$ |
| $(1+x+x^2)^5$ | $1+5x+15x^2+30x^3+45x^4+51x^5+45x^6+30x^7+15x^8+5x^9+x^{10}$ |

*GegenbauerC[n, –t, –1/2]*

Algebraic representations of the patterns on the facing page. The coefficient of $x^n$ on each row gives the value of each square. These coefficients can also be obtained from the formulas in terms of *Binomial* and *GegenbauerC* given. A particular square is colored black if its value $a$ is odd. This can be determined either from *Mod[a, 2]* or equivalently from $(1-(-1)^a)/2$ or $Sin[\pi\, a/2]^2$. The succession of polynomials above can be obtained by expanding the generating functions $1/(1-(1+x)\,y)$ and $1/(1-(1+x+x^2)\,y)$. *Binomial[m, n]* is the ordinary binomial coefficient $m!/(n!\,(m-n)!)$. *GegenbauerC* is a so-called orthogonal polynomial—a higher mathematical function.

So what about the bottom picture on the facing page? Much as in the top picture numbers can be assigned to each square, but now these numbers are computed by successively adding together triples rather



Nested patterns constructed using arithmetic operations. The example at the top is Pascal's triangle, formed by making each number be the sum of the numbers immediately to its left and right on the row above. The example at the bottom is a generalization of Pascal's triangle in which each number is the sum of the numbers above it and to its left and right on the row above. In both cases squares are colored black when the numbers that appear in them are odd. The limiting arrangements of colors correspond to nested patterns. For the top picture the pattern is what would be generated by an additive cellular automaton following rule 90; for the bottom picture it is what would be generated by one following rule 150. The numbers in the top picture are binomial coefficients; those in the bottom picture are particular trinomial coefficients.

than pairs. And once again the numbers appear as coefficients, but now in the expansion of powers of $1 + x + x^2$ rather than of $1 + x$.

So is there an explicit formula for these coefficients? If one restricts oneself to a fixed number of elementary mathematical

functions together with factorials and multinomial coefficients then it appears that there is not. But if one also allows higher mathematical functions then it turns out that such a formula can in fact be found: as indicated in the table above each coefficient is given by a particular value of a so-called Gegenbauer or ultraspherical function.

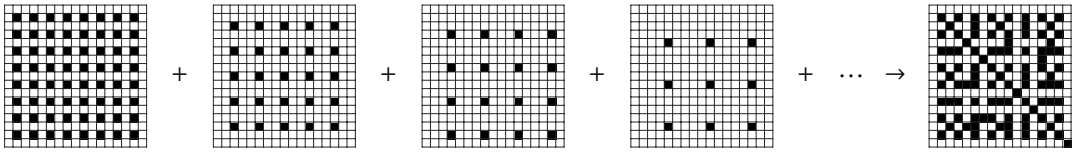
So what about other nested patterns? Both of the patterns shown on the previous page are rather special in that as well as being generated by substitution systems they can also be produced one row at a time by the evolution of one-dimensional cellular automata with simple additive rules. And in fact the approaches used above can be viewed as direct generalizations of such additive rules to the domain of ordinary numbers.

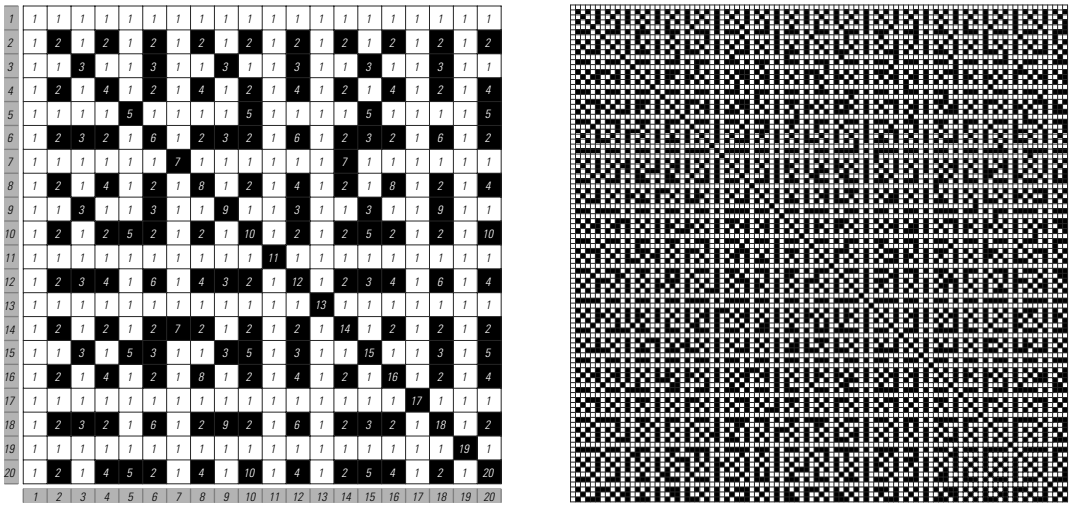
For a few other nested patterns there exist fairly simple connections with additive cellular automata and similar systems—though usually in more dimensions or with more neighbors. But for most nested patterns there seems to be no obvious way to relate them to ordinary mathematical functions. Nevertheless, despite this, it is my guess that in the end it will in fact turn out to be possible to get a formula for any nested pattern in terms of suitably generalized hypergeometric functions, or perhaps other functions that are direct generalizations of ones used in traditional mathematics.

Yet given how simple and regular nested patterns tend to look it may come as something of a surprise that it should be so difficult to represent them as traditional mathematical formulas. And certainly if this example is anything to go by, it begins to seem unlikely that the more complex kinds of patterns that we have seen so many times in this book could ever realistically be represented by such formulas.

But it turns out that there are at least some cases where traditional mathematical formulas can be found even though to the eye or with respect to other methods of perception and analysis a pattern may seem highly complex.

The picture at the top of the facing page is one example. A pattern is built up by superimposing a sequence of repetitive grids, and to the eye this pattern seems highly complex. But in fact there is a simple formula for the color of each square: given the largest factor in common between the





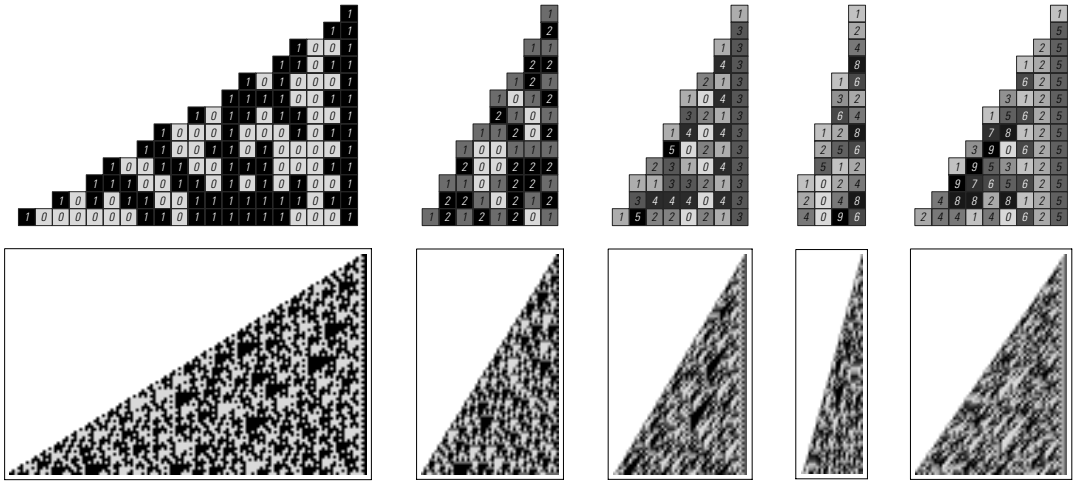
An example of a pattern that looks complex, but can nevertheless still be represented by a simple mathematical formula. Given the horizontal and vertical positions $x$ and $y$ a square is white when $GCD[x, y] = 1$ and is black otherwise. The condition $GCD[x, y] = 1$ is equivalent to the statement that $x$ and $y$ are relatively prime, or that no reduction is required to bring the fraction $x/y$ to lowest terms. It can be shown that if the pattern is extended sufficiently far, then any possible local arrangement of black squares will eventually appear—though not necessarily with equal frequency.

numbers that specify the horizontal and vertical positions of the square, the square is white whenever this factor is 1, and is black otherwise.

So what about systems like cellular automata that have definite rules for evolution? Are there ever cases in which patterns generated by such systems seem complex to the eye but can in fact be described by simple mathematical formulas?

I know of one class of examples where this happens, illustrated in the pictures on the next page. The idea is to set up a row of cells corresponding to the digits of a number in a certain base, and then at each step to multiply this number by some fixed factor.

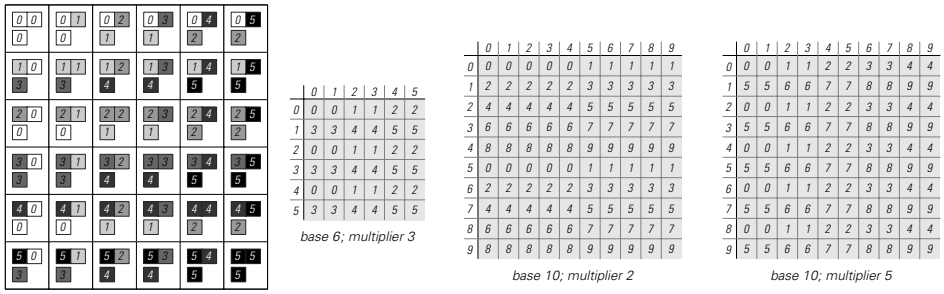
Such a system has many features immediately reminiscent of a cellular automaton. But at least in the case of multiplication by 3 in



base 2; multiplier 3 | base 3; multiplier 2 | base 6; multiplier 3 | base 10; multiplier 2 | base 10; multiplier 5

Patterns of digits in various bases generated by successive multiplication by a fixed factor. Such systems were discussed on page 120. With multiplier $m$ row $t$ corresponds to the power $m^t$. The value of the cell at position $n$ from the end of row $t$ is thus the $n^{\text{th}}$ digit of $m^t$, or *Mod[Quotient[$m^t$, $k^n$], k]*. Despite the apparent complexity of the patterns, a fairly simple mathematical formula thus exists for the color of each square they contain.

base 2, the presence of carry digits in the multiplication process makes the system not quite an ordinary cellular automaton. It turns out, however, that multiplication by 3 in base 6, or by 2 or 5 in base 10, never leads to carry digits, with the result that in such cases the system can be thought of as following a purely local cellular automaton rule of the kind illustrated below.



base 6; multiplier 3 | base 10; multiplier 2 | base 10; multiplier 5

Cellular automaton rules equivalent to multiplication of digit sequences in various bases. The left part of the picture shows the explicit form of the rule for base 6 and multiplier 3. The arrays of numbers summarize the rule for this case and other cases. Note that only certain specific choices of base and multiplier lead to ordinary cellular automata; with other choices there are carries that propagate arbitrarily far. (See page 661.)

As the pictures at the top of the facing page demonstrate, the overall patterns produced in all cases tend to look complex, and in many respects random. But the crucial point is that because of the way the system was constructed there is nevertheless a simple formula for the color of each cell: it is given just by a particular digit in the number obtained by raising the multiplier to a power equal to the number of steps. So despite their apparent complexity, all the patterns on the facing page can in effect be described by simple traditional mathematical formulas.

But if one thinks about actually using such formulas one might at first wonder what good they really are. For if one was to work out the value of a power $m^t$ by explicitly performing $t$ multiplications, this would be very similar to explicitly following $t$ steps of cellular automaton evolution. But the point is that because of certain mathematical features of powers it turns out to be possible—as indicated in the table below—to find $m^t$ with many fewer than $t$ operations; indeed, one or two operations for every base 2 digit in $t$ is always for example sufficient.

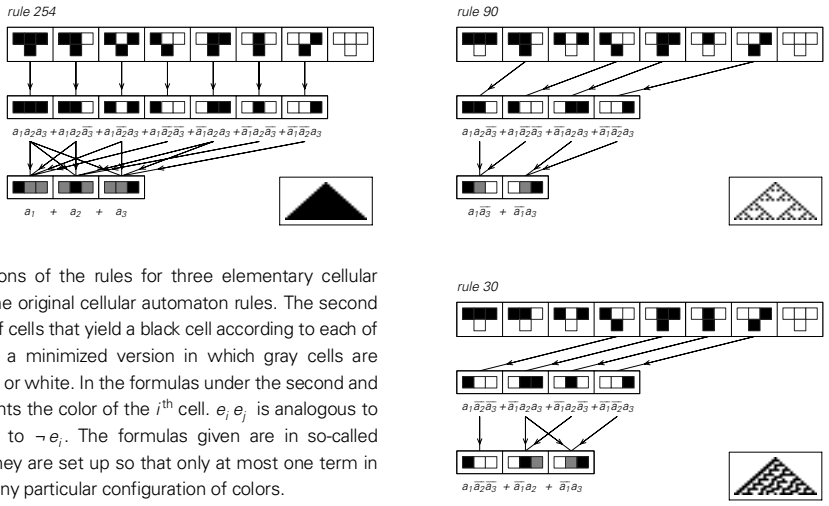
| | | |
|---|---|---|
| $m^1$ | $m$ | $m$ |
| $m^2$ | $m \times m$ | $m^2$ |
| $m^3$ | $m \times m \times m$ | $m^2 \times m$ |
| $m^4$ | $m \times m \times m \times m$ | $(m^2)^2$ |
| $m^5$ | $m \times m \times m \times m \times m$ | $(m^2)^2 \times m$ |
| $m^6$ | $m \times m \times m \times m \times m \times m$ | $(m^2 \times m)^2$ |
| $m^7$ | $m \times m \times m \times m \times m \times m \times m$ | $(m^2 \times m)^2 \times m$ |
| $m^8$ | $m \times m \times m \times m \times m \times m \times m \times m$ | $((m^2)^2)^2$ |
| $m^9$ | $m \times m \times m \times m \times m \times m \times m \times m \times m$ | $((m^2)^2)^2 \times m$ |
| $m^{10}$ | $m \times m \times m \times m \times m \times m \times m \times m \times m \times m$ | $((m^2)^2 \times m)^2$ |

Examples of how powers can be computed more efficiently than by successive multiplications. In the cases shown, the choice of whether to square or multiply by an additional factor of $m$ at each step in computing $m^t$ is made on the basis of the successive digits in the base 2 representation of the number $t$.

So what about other patterns produced by cellular automata and similar systems? Is it possible that in the end all such patterns could just be described by simple mathematical formulas? I do not think so. In fact, as I will argue in Chapter 12, my strong belief is that in the vast majority of cases it will be impossible for quite fundamental reasons to find any

such simple formula. But even though no simple formula may exist, it is still always in principle possible to represent the outcome of any process of cellular automaton evolution by at least some kind of formula.

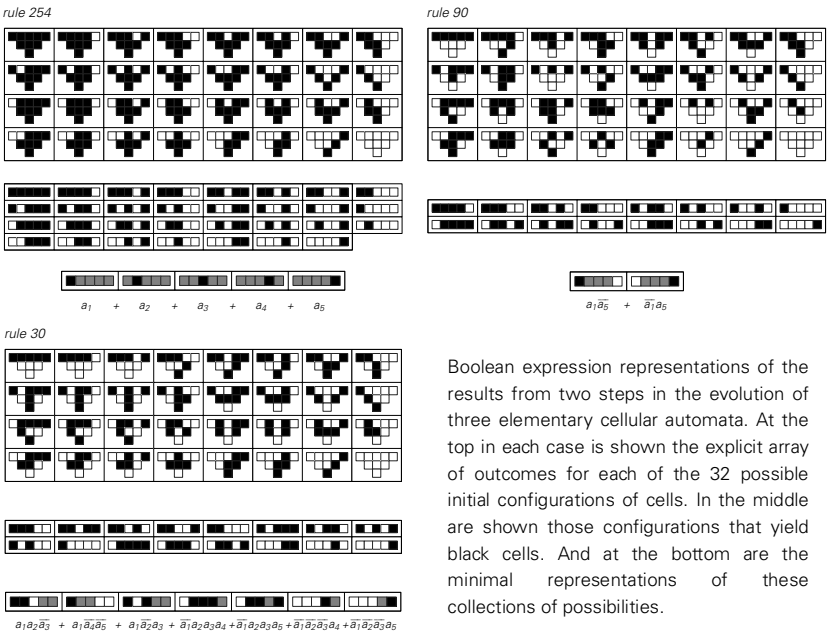
The picture below shows how this can be done for a single step in the evolution of three elementary cellular automata. The basic idea is to translate the rule for a given cellular automaton into a formula that depends on three variables $a_1$, $a_2$ and $a_3$ whose values correspond to the colors of the three initial cells. The formula consists of a sum of terms, with each term being zero unless the colors of the three cells match a situation in which the rule yields a black cell.




Boolean expression representations of the rules for three elementary cellular automata. The first row shows the original cellular automaton rules. The second row shows those combinations of cells that yield a black cell according to each of the rules. The third row shows a minimized version in which gray cells are introduced to indicate either black or white. In the formulas under the second and third rows the variable $a_i$ represents the color of the $i^{\text{th}}$ cell. $e_i\, e_j$ is analogous to $e_i \wedge e_j$, $e_i + e_j$ to $e_i \vee e_j$, and $\overline{e_i}$ to $\neg e_i$. The formulas given are in so-called disjunctive normal form (DNF). They are set up so that only at most one term in each formula is ever relevant for any particular configuration of colors.

In the first instance, each term can be set up to correspond directly to one of the cases in the original rule. But in general this will lead to a more complicated formula than is necessary. For as the picture demonstrates, it is often possible to combine several cases into one term by ignoring the values of some of the variables.

The picture at the top of the facing page shows what happens if one considers two steps of cellular automaton evolution. There are now altogether five variables, but at least for rules like rules 254 and 90 the individual terms end up not depending on most of these variables.




Boolean expression representations of the results from two steps in the evolution of three elementary cellular automata. At the top in each case is shown the explicit array of outcomes for each of the 32 possible initial configurations of cells. In the middle are shown those configurations that yield black cells. And at the bottom are the minimal representations of these collections of possibilities.

So what happens if one considers more steps? As the pictures on the next page demonstrate, rules like 254 and 90 that have fairly simple behavior lead to formulas that stay fairly simple. But for rule 30 the formulas rapidly get much more complicated.

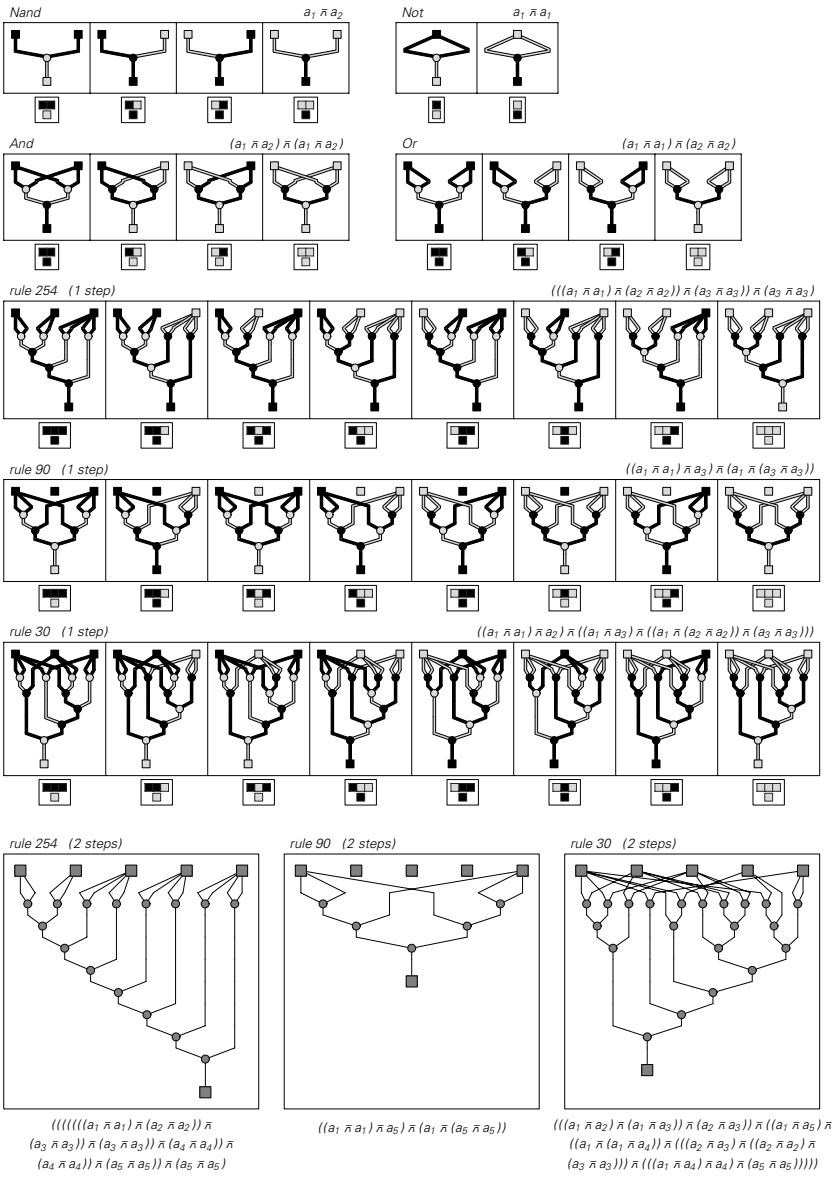
So this strongly suggests that no simple formula exists—at least of the type used here—that can describe patterns generated by any significant number of steps of evolution in a system like rule 30.

But what about formulas of other types? The formulas we have used so far can be thought of as always consisting of sums of products of variables. But what if we allow formulas with more general structure, not just two fixed levels of operations?

It turns out that any rule for blocks of black and white cells can be represented as some combination of just a single type of operation—for example a so-called NAND function of the kind often used in digital electronics. And given this, one can imagine finding for any particular rule the formula that involves the smallest number of NAND functions.


Minimal Boolean expression representations for the results of steps 1 through 5 in the evolution of three elementary cellular automata. Both rules 254 and 90 have fairly simple overall behavior, and yield comparatively small Boolean expressions. Rule 30 has much more complicated behavior and yields Boolean expressions whose size grows rapidly from one step to the next. (For steps 1 through 6, the expressions involve 3, 7, 17, 41, 102 and 261 terms respectively.) In each case the Boolean expressions given are the smallest possible in the disjunctive normal form (DNF) used.

The picture below shows some examples of the results. And once again what we see is that for rules with fairly simple behavior the formulas are usually fairly simple. But in cases like rule 30, the formulas one gets are already quite complicated even after just two steps.




Minimal representations in terms of NAND functions of the first two steps in the evolution of the same cellular automata as on the facing page. In each case, the network and formula shown are ones that involve the absolute minimum number of operations. Finding these effectively required searching through billions of possibilities. The picture at the top left shows the action of a single NAND function. The next three pictures show how the operations used in DNF formulas can be built up from NANDs.

So even if one allows rather general structure, the evidence is that in the end there is no way to set up any simple formula that will describe the outcome of evolution for a system like rule 30.

And even if one settles for complicated formulas, just finding the least complicated one in a particular case rapidly becomes extremely difficult. Indeed, for formulas of the type shown on page 618 the difficulty can already perhaps double at each step. And for the more general formulas shown on the previous page it may increase by a factor that is itself almost exponential at each step.

So what this means is that just like for every other method of analysis that we have considered, we have little choice but to conclude that traditional mathematics and mathematical formulas cannot in the end realistically be expected to tell us very much about patterns generated by systems like rule 30.

## Human Thinking

When we are presented with new data one thing we can always do is just apply our general powers of human thinking to it. And certainly this allows us with rather modest effort to do quite well in handling all sorts of data that we choose to interact with in everyday life. But what about data generated by the kinds of systems that I have discussed in this book? How does general human thinking do with this?

There are definitely some limitations, since after all, if general human thinking could easily find simple descriptions of, for example, all the various pictures in this book, then we would never have considered any of them complex.

One might in the past have assumed that if a simple description existed of some piece of data, then with appropriate thinking and intelligence it would usually not be too difficult to find it. But what the results in this book establish is that in fact this is far from true. For in the course of this book we have seen a great many systems whose underlying rules are extremely simple, yet whose overall behavior is sufficiently complex that even by thinking quite hard we cannot recognize its simple origins.

Usually a small amount of thinking allows us to identify at least some regularities. But typically these regularities are ones that can also be found quite easily by many of the standard methods of perception and analysis discussed earlier in this chapter.

So what then does human thinking in the end have to contribute? The most obvious way in which it stands out from other methods of perception and analysis is in its large-scale use of memory.

For all the other methods that we have discussed effectively operate by taking each new piece of data and separately applying some fixed procedure to it. But in human thinking we routinely make use of the huge amount of memory that we have built up from being exposed to billions of previous pieces of data.

And sometimes the results can be quite impressive. For it is quite common to find that even though no other method has much to say about a particular piece of data, we can immediately come up with a description for it by remembering some similar piece of data that we have encountered before.

And thus, for example, having myself seen thousands of pictures produced by cellular automata, I can recognize immediately from memory almost any pattern generated by any of the elementary rules—even though none of the other methods of perception and analysis can get very far whenever such patterns are at all complex.

But insofar as there is sophistication in what can be done with human memory, does this sophistication come merely from the experiences that are stored in memory, or somehow from the actual mechanism of memory itself?

The idea of storing large amounts of data and retrieving it according to various criteria is certainly quite familiar from databases in practical computing. But there is at least one important difference between the way typical databases operate, and the way human memory operates. For in a standard database one tends to be able to find only data that meets some precise specification, such as containing an exact match to a particular string of text. Yet with human memory we routinely seem to be able to retrieve data on the basis of much more general notions of similarity.

In general, if one wants to find a piece of data that has a certain property—either exact or approximate—then one way to do this is just to scan all the pieces of data that one has stored, and test each of them in turn. But even if one does all sorts of parallel processing this approach presumably in the end becomes quite impractical.

So what can one then do? In the case of exact matches there are a couple of approaches that are widely used in practice.

Probably the most familiar is what is done in typical dictionaries: all the entries are arranged in alphabetical order, so that when one looks something up one does not need to scan every single entry but instead one can quickly home in on just the entry one wants.

Practical database systems almost universally use a slightly more efficient scheme known as hashing. The basic idea is to have some definite procedure that takes any word or other piece of data and derives from it a so-called hash code which is used to determine where the data will be stored. And the point is that if one is looking for a particular piece of data, one can then apply this same procedure to that data, get the hash code for the data, and immediately determine where the data would have been stored.

But to make this work, does one need a complex hashing procedure that is carefully tuned to the particular kind of data one is dealing with? It turns out that one does not. And in fact, all that is really necessary is that the hashing procedure generate enough randomness that even though there may be regularities in the original data, the hash codes that are produced still end up being distributed roughly uniformly across all possibilities.

And as one might expect from the results in this book, it is easy to achieve this even with extremely simple programs—either based on numbers, as in most practical database systems, or based on systems like cellular automata.

So what this means is that regardless of what kind of data one is storing, it takes only a very simple program to set up a hashing scheme that lets one retrieve pieces of data very efficiently. And I suspect that at least some aspects of this kind of mechanism are involved in the operation of human memory.

But what about the fact that we routinely retrieve from our memory not just data that matches exactly, but also data that is merely similar? Ordinary hashing would not let us do this. For a hashing procedure will normally put different pieces of data at quite different locations—even if the pieces of data happen in some sense to be similar.

So is it possible to set up forms of hashing that will in fact keep similar pieces of data together? In a sense what one needs is a hashing procedure in which the hash codes that are generated depend only on features of the data that really make a difference, and not on others.

One practical example where this is done is a simple procedure often used for looking up names by sound rather than spelling. In its typical form this procedure works by dropping all vowels and grouping together letters like "d" and "t" that sound similar, with the result that at least in some approximation the only features that are kept are ones that make a difference in the way a word sounds.

So how can one achieve this in general?

In many respects one of the primary goals of all forms of perception and analysis is precisely to pick out those features of data that are considered relevant, and to discard all others.

And so, as we discussed earlier in this chapter, the human visual system, for example, appears to be based on having nerve cells that respond only to certain specific features of images. And this means that if one looks only at the output from these nerve cells, then one gets a representation of visual images in which two images that differ only in certain kinds of details will be assigned the same representation.

So if it is a representation like this that is used as the basis for storing data in memory, the result is that one will readily be able to retrieve not only data that matches exactly, but also data that is merely similar enough to have the same representation.

In actual brains it is fairly clear that input received by all the various sensory systems is first processed by assemblies of nerve cells that in effect extract certain specific features. And it seems likely that especially in lower organisms it is often representations formed quite directly from such features that are what is stored in memory.

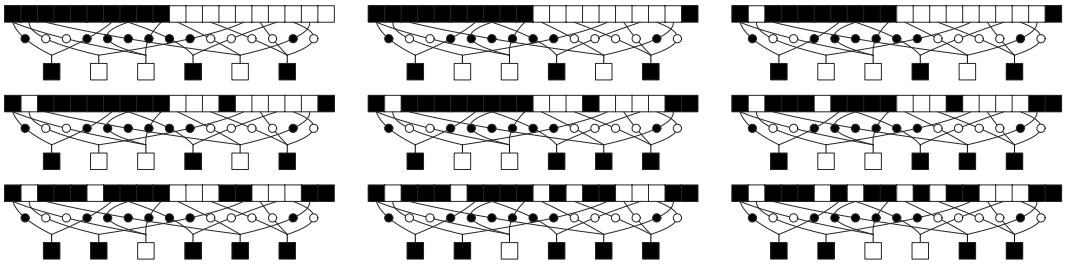
But at least in humans there is presumably more going on. For it is quite common that we can immediately recognize that we have encountered some particular object before even if it is superficially presented in a quite different way. And what this suggests is that quite different patterns of raw data from our sensory systems can at least in some cases still lead to essentially the same representation in memory.

So how might this be achieved? One possibility is that our brains might be set up to extract certain specific high-level features—such as, say, topological structure in three-dimensional space—that happen to successfully characterize particular kinds of objects that we traditionally deal with.

But my strong suspicion is that in fact there is some much simpler and more general mechanism at work, that operates essentially just at the level of arbitrary data elements, without any direct reference to the origin or meaning of these data elements.

And one can imagine quite a few ways that such a mechanism could potentially be set up with nerve cells.

One step in a particularly simple scheme is illustrated in the picture below. The basic idea is to have a sequence of layers of nerve cells—much as one knows exist in the brain—with each cell in each successive layer responding only if the inputs it gets from some fixed random set of cells in the layer above form some definite pattern.



One step in a very simple model of the way hash codes for arbitrary data might be generated by layers of nerve cells in the brain. The response of a single layer of idealized nerve cells to a sequence of progressively different inputs is shown. Each nerve cell fires and yields black output only if the inputs it gets from certain fixed positions match a particular template. The sequence of outputs from all the nerve cells can be used as a hash code, whose value tends to be the same for inputs that differ only by small changes.

In a sense this is a straightforward generalization of the scheme for visual perception that we discussed earlier in this chapter. But the point is that with such a setup detailed changes in the input to the first layer of cells only rarely end up having an effect on output from the last layer of cells.

It is not difficult to find systems in which different inputs often yield the same output. In fact, this is the essence of the very general phenomenon of attractors that we discussed in Chapter 6—and it is seen in the vast majority of cellular automata, and in fact in almost any kind of system that follows definite rules.

But what is somewhat special about the setup above is that inputs which yield the same output tend to be ones that might reasonably be considered similar, while inputs that yield different outputs tend to be significantly different.

And thus, for example, a change in a single input cell typically will not have a high probability of affecting the output, while a change in a large fraction of the input cells will.

So quite independent of precisely which features of the original data correspond to which input cells, this basic mechanism provides a simple way to get a representation—and thus a hash code—that will tend to be the same for pieces of data that somehow have enough features that are similar.

So how would such a representation in the end be used? In a scheme like the one above the output cells would presumably be connected to cells that actually perform actions of some kind—perhaps causing muscles to move, or perhaps just providing inputs to further nerve cells.

But so where in all of this would the actual content of our memory reside? Almost certainly at some level it is encoded in the details of connections between nerve cells.

But how then might such details get set up?

There is evidence that permanent changes can be produced in individual nerve cells as a result of the behavior of nerve cells around them. And as data gets received by the brain such changes presumably do occur at least in some cells. But if one looks, say, at nerve cells involved in the early stages of the visual system, then once the brain has matured past some point these never seem to change their properties

much. And quite probably the same is true of many nerve cells involved in the general process of doing the analog of producing hash codes.

The reason for such a lack of change could conceivably be simply that at the relevant level the overall properties of the stream of data corresponding to typical experience remain fairly constant. But it might also be that if one expects to retrieve elements of memory reliably then there is no choice but to set things up so that the hashing procedure one uses always stays essentially the same.

And if there is a fixed such scheme, then this implies that while certain similarities between pieces of data will immediately be recognized, others will not.

So how does this compare to what we know of actual human memory? There are many kinds of similarities that we recognize quite effortlessly. But there are also ones that we do not. And thus, for example, given a somewhat complicated visual image—say of a face or a cellular automaton pattern—we can often not even immediately recognize similarity to the same image turned upside-down.

So are such limitations in the end intrinsic to the underlying mechanism of human memory, or do they somehow merely reflect characteristics of the memory that we happen to build up from our typical actual experience of the world?

My guess is that it is to some extent a mixture. But insofar as more important limitations tend to be the result of quite low-level aspects of our memory system it seems likely that even if these aspects could in principle be changed it would in practice be essentially impossible to do so. For the low levels of our memory system are exposed to an immense stream of data. And so to cause any substantial change one would presumably have to insert a comparable amount of data with the special properties one wants. But for a human interacting with anything like a normal environment this would in practice be absolutely impossible.

So in the end I strongly suspect that the basic rules by which human memory operates can almost always be viewed as being essentially fixed—and, I believe, fairly simple.

But what about the whole process of human thinking? What does it ultimately involve? My strong suspicion is that the use of memory is what in fact underlies almost every major aspect of human thinking.

Capabilities like generalization, analogy and intuition immediately seem very closely related to the ability to retrieve data from memory on the basis of similarity. But what about capabilities like logical reasoning? Do these perhaps correspond to a higher-level type of human thinking?

In the past it was often thought that logic might be an appropriate idealization for all of human thinking. And largely as a result of this, practical computer systems have always treated logic as something quite fundamental. But it is my strong suspicion that in fact logic is very far from fundamental, particularly in human thinking.

For among other things, whereas in the process of thinking we routinely manage to retrieve remarkable connections almost instantaneously from memory, we tend to be able to carry out logical reasoning only by laboriously going from one step to the next. And my strong suspicion is that when we do this we are in effect again just using memory, and retrieving patterns of logical argument that we have learned from experience.

In modern times computer languages have often been thought of as providing precise ways to represent processes that might otherwise be carried out by human thinking. But it turns out that almost all of the major languages in use today are based on setting up procedures that are in essence direct analogs of step-by-step logical arguments.

As it happens, however, one notable exception is *Mathematica*. And indeed, in designing *Mathematica*, I specifically tried to imitate the way that humans seem to think about many kinds of computations. And the structure that I ended up coming up with for *Mathematica* can be viewed as being not unlike a precise idealization of the operation of human memory.

For at the core of *Mathematica* is the notion of storing collections of rules in which each rule specifies how to transform all pieces of data that are similar enough to match a single *Mathematica* pattern. And the success of *Mathematica* provides considerable evidence for the power of this kind of approach.

But ultimately—like other computer languages—*Mathematica* tends to be concerned mostly with setting up fairly short specifications for quite definite computations. Yet in everyday human thinking we seem instead to use vast amounts of stored data to perform tasks whose definitions and objectives are often quite vague.

There has in the past been a great tendency to assume that given all its apparent complexity, human thinking must somehow be an altogether fundamentally complex process, not amenable at any level to simple explanation or meaningful theory.

But from the discoveries in this book we now know that highly complex behavior can in fact arise even from very simple basic rules. And from this it immediately becomes conceivable that there could in reality be quite simple mechanisms that underlie human thinking.

Certainly there are many complicated details to the construction of the brain, and no doubt there are specific aspects of human thinking that depend on some of these details. But I strongly suspect that there is a definite core to the phenomenon of human thinking that is largely independent of such details—and that will in the end turn out to be based on rules that are rather simple.

So how will we be able to tell if this is in fact the case? Detailed direct studies of the brain and its operation may give some clues. But my guess is that the only way that really convincing evidence will be obtained is if actual technological systems are constructed that can successfully be seen to emulate human thinking.

And indeed as of now our experience with practical computing provides rather little encouragement that this will ever be possible. There are certainly some tasks—such as playing chess or doing algebra—that at one time were considered indicative of human-like thinking, but which are now routinely done by computer. Yet when it comes to seemingly much more mundane and everyday types of thinking the computers and programs that exist at present tend to be almost farcically inadequate.

So why have we not done better? No doubt part of the answer has to do with various practicalities of computers and storage systems. But a more important part, I suspect, has to do with issues of methodology.

For it has almost always been assumed that to emulate in any generality a process as sophisticated as human thinking would necessarily require an extremely complicated system. So what has mostly been done is to try to construct systems that perform only rather specific tasks.

But then in order to be sure that the appropriate tasks will actually be performed the systems tend to be set up—as in traditional engineering—so that their behavior can readily be foreseen, typically by standard mathematical or logical methods. And what this almost invariably means is that their behavior is forced to be fairly simple. Indeed, even when the systems are set up with some ability to learn they usually tend to act—much like the robots of classical fiction—with far too much simplicity and predictability to correspond to realistic typical human thinking.

So on the basis of traditional intuition, one might then assume that the way to solve this problem must be to use systems with more complicated underlying rules, perhaps more closely based on details of human psychology or neurophysiology. But from the discoveries in this book we know that this is not the case, and that in fact very simple rules are quite sufficient to produce highly complex behavior.

Nevertheless, if one maintains the goal of performing specific well-defined tasks, there may still be a problem. For insofar as the behavior that one gets is complex, it will usually be difficult to direct it to specific tasks—an issue rather familiar from dealing with actual humans. So what this means is that most likely it will at some level be much easier to reproduce general human-like thinking than to set up some special version of human-like thinking only for specific tasks.

And it is in the end my strong suspicion that most of the core processes needed for general human-like thinking will be able to be implemented with rather simple rules.

But a crucial point is that on their own such processes will most likely not be sufficient to create a system that one would readily recognize as exhibiting human-like thinking. For in order to be able to relate in a meaningful way to actual humans, the system would almost certainly have to have built up a human-like base of experience.

No doubt as a practical matter this could to some extent be done just by large-scale recording of experiences of actual humans. But it seems not unlikely that to get a sufficiently accurate experience base, the system would itself have to interact with the world in very much the same way as an actual human—and so would have to have elements that emulate many elaborate details of human biological and other structure.

Once one has an explicit system that successfully emulates human thinking, however, one can imagine progressively removing some of this complexity, and seeing just which features of human thinking end up being preserved.

So what about human language, for example? Is this purely learned from the details of human experience? Or are there features of it that reflect more fundamental aspects of human thinking?

When one learns a language—at least as a young child—one implicitly tends to deduce simple grammatical rules that are in effect specific generalizations of examples one has encountered. And I suspect that in doing this the types of generalizations that one makes are essentially those that correspond to the types of similarities that one readily recognizes in retrieving data from memory.

Actual human languages normally have many exceptions to any simple grammatical rules. And it seems that with sufficient effort we can in fact learn languages with almost any structure. But the fact that most modern computer languages are specifically set up to follow simple grammatical rules seems to make their structures particularly easy for us to learn—perhaps because they fit in well with low-level processes of human thinking.

But to what extent is the notion of a language even ultimately necessary in a system that does human-like thinking? Certainly in actual humans, languages seem to be crucial for communication. But one might imagine that if the underlying details of different individuals from some class of systems were sufficiently identical then communication could instead be achieved just by directly transferring low-level patterns of activity. My guess, however, is that as soon as the experiences of different individuals become different, this will not

work, and that therefore some form of general intermediate representation or language will be required.

But does one really need a language that has the kind of sequential grammatical structure of ordinary human language? Graphical user interfaces for computer systems certainly often use somewhat different schemes. And in simple situations these can work well. But my uniform experience has been that if one wants to specify processes of any significant complexity in a fashion that can reasonably be understood then the only realistic way to do this is to use a language—like *Mathematica*—that has essentially an ordinary sequential grammatical structure.

Quite why this is I am not certain. Perhaps it is merely a consequence of our familiarity with traditional human languages. Or perhaps it is a consequence of our apparent ability to pay attention only to one thing at a time. But I would not be surprised if in the end it is a reflection of fairly fundamental features of human thinking.

And indeed our difficulty in thinking about many of the patterns produced by systems in this book may be not unrelated. For while ordinary human language has little trouble describing repetitive and even nested patterns, it seems to be able to do very little with more complex patterns—which is in a sense why this book, for example, depends so heavily on visual presentation.

At the outset, one might have imagined that human thinking must involve fundamentally special processes, utterly different from all other processes that we have discussed. But just as it has become clear over the past few centuries that the basic physical constituents of human beings are not particularly special, so also—especially after the discoveries in this book—I am quite certain that in the end there will turn out to be nothing particularly special about the basic processes that are involved in human thinking.

And indeed, my strong suspicion is that despite the apparent sophistication of human thinking most of the important processes that underlie it are actually very simple—much like the processes that seem to be involved in all the other kinds of perception and analysis that we have discussed in this chapter.

## Higher Forms of Perception and Analysis

In the course of this chapter we have discussed in turn each of the major methods of perception and analysis that we in practice use. And if our goal is to understand the actual experience that we get of the world then there is no reason to go further. But as a matter of principle one can ask whether the methods of perception and analysis that we have discussed in a sense cover what is ultimately possible—or whether instead there are higher and fundamentally more powerful forms of perception and analysis that for some reason we do not at present use.

As we discussed early in this chapter, any method of perception or analysis can at some level be viewed as a way of trying to find simple descriptions for pieces of data. And what we might have assumed in the past is that if a piece of data could be generated from a sufficiently simple description then the data itself would necessarily seem to us quite simple—and would therefore have many regularities that could be recognized by our standard methods of perception and analysis.

But one of the central discoveries of this book is that this is far from true—and that actually it is rather common for rules that have extremely simple descriptions to give rise to data that is highly complex, and that has no regularities that can be recognized by any of our standard methods.

But as we discussed earlier in this chapter the fact that a simple rule can ultimately be responsible for such data means that at some level the data must contain regularities. So the point is that these regularities are just not ones that can be detected by our standard methods of perception and analysis.

Yet the fact that there are in the end regularities means that at least in principle there could exist higher forms of perception and analysis that would succeed in recognizing them.

So might one day some new method of perception and analysis be invented that would in a sense manage to recognize all possible regularities, and thus be able to tell immediately if any particular piece of data could be generated from any kind of simple description?

My strong belief—as I will argue in Chapter 12—is that at least in complete generality this will never be possible. But that does not mean that there cannot exist higher forms of perception and analysis that succeed in recognizing at least some regularities that our existing methods do not.

The results of this chapter, however, might seem to provide some circumstantial evidence that in practice even this might not be possible. For in the course of the chapter we have discussed a whole range of different kinds of perception and analysis, yet in essentially all cases we have found that the overall capabilities they exhibit are rather similar. Most of them, for example, recognize repetition, and some also recognize nesting. But almost none recognize anything more complex.

So what this perhaps suggests is that in the end there might be only certain specific capabilities that can be realized in practical methods of perception and analysis. And certainly it seems not inconceivable that there could be a fundamental result that the only kinds of regularities that both occur frequently in actual systems and can be recognized quickly enough to provide a basis for practical methods of perception and analysis are ones like repetition and nesting.

But there is another possible explanation for what we have seen in this chapter: perhaps it is just that we, as humans, are always very narrow in the methods of perception and analysis that we use. For certainly it is remarkable that none of the methods that we normally use ever in the end seem to manage to get much further than we can already get with our own built-in powers of perception. And what this perhaps suggests is that we choose the methods we use to be essentially those that pick out only regularities with which we are somehow already very familiar from our own built-in powers of perception.

For there is no difficulty in principle in constructing procedures that have capabilities very different from those of our standard methods of perception and analysis. Indeed, as one example, one could imagine just enumerating all possible simple descriptions of some particular type, and then testing in each case to see whether what one gets matches a piece of data that one has.

And in some specific cases, this might well succeed in finding extremely simple descriptions for the data. But to use such a method in

any generality almost inevitably requires computational resources far greater than one would normally consider reasonable in a practical method of perception or analysis.

And in fact there is really no reason to consider such a sophisticated procedure. For in a sense any program—including one that is very simple and runs very quickly—can be thought of as implementing a method of perception or analysis. For if one gives a piece of data as the input to the program, then the output one gets—whatever it may be—can be viewed as corresponding to some kind of description of the data.

But the problem is that under most circumstances this description will not be particularly useful. And indeed what typically seems to be necessary to make it useful is that somehow one is already familiar with similar descriptions, and knows their significance.

A description based on output from a cellular automaton rule that one has never seen before is thus for example not likely to be useful. But a description that picks out a feature like repetition that is already very familiar to us will typically be much more useful.

And potentially therefore our lack of higher forms of perception and analysis might in the end have nothing to do with any difficulty in implementing such forms, but instead may just be a reflection of the fact that we only have enough context to make descriptions of data useful when these descriptions are fairly close to the ones we get from our own built-in human methods of perception.

But why is it then that these methods themselves are not more powerful? After all, one might think that biological evolution would inevitably have made us as good as possible at handling data associated with any of the systems that we commonly encounter in nature.

Yet as we have seen in this book almost whenever there is significant complexity our powers of human perception end up being far from adequate to find any kind of minimal summaries of data.

And with the traditional view that biological evolution is somehow a process of infinite power this seems to leave one little choice but to conclude that there must be fundamental limitations on possible methods of perception that can be useful.

One might imagine perhaps that while there could in principle be methods of perception that would recognize features beyond, say, repetition and nesting, any single such feature might never occur in a sufficiently wide range of systems to make its recognition generally useful to a biological organism.

But as of now I do not know of any fundamental reason why this might be so, and following my arguments in Chapter 8 I would not be at all surprised if the process of biological evolution had simply missed even methods of perception that are, in some sense, fairly obvious.

So what about an extraterrestrial intelligence? Free from any effects of terrestrial biological evolution might it have developed all sorts of higher forms of perception and analysis?

Of course we have no direct information on this. But the very fact that we have so far failed to discover any evidence for extraterrestrial intelligence may itself conceivably already be a sign that higher forms of perception and analysis may be in use.

For as I will discuss in Chapter 12 it seems far from inconceivable that some of the extraterrestrial radio and other signals that we pick up and assume to be random noise could in fact be meaningful messages—but just encoded in a way that can be recognized only by higher forms of perception and analysis than those we have so far applied to them.

Yet whether or not this is so, the capabilities of extraterrestrial intelligence are not in the end directly relevant to an understanding of our own experience of the world. In the future we may well manage to use higher forms of perception and analysis, and as a result our experience of the world will change—no doubt along with certain aspects of our science and mathematics. But for now it is the kinds of methods of perception and analysis that we have discussed in most of this chapter that must form the basis for the conclusions we make about the world.