EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

SECTION 12.6

Computational Irreducibility

analysis used to study them must be processes based on natural laws. But at least in the recent history of science it has normally been assumed that the evolution of typical systems in nature is somehow much less sophisticated a process than perception and analysis.

Yet what the Principle of Computational Equivalence now asserts is that this is not the case, and that once a rather low threshold has been reached, any real system must exhibit essentially the same level of computational sophistication. So this means that observers will tend to be computationally equivalent to the systems they observe—with the inevitable consequence that they will consider the behavior of such systems complex.

So in the end the fact that we see so much complexity can be attributed quite directly to the Principle of Computational Equivalence, and to the fact that so many of the systems we encounter in practice turn out to be computationally equivalent.

Computational Irreducibility

When viewed in computational terms most of the great historical triumphs of theoretical science turn out to be remarkably similar in their basic character. For at some level almost all of them are based on finding ways to reduce the amount of computational work that has to be done in order to predict how some particular system will behave.

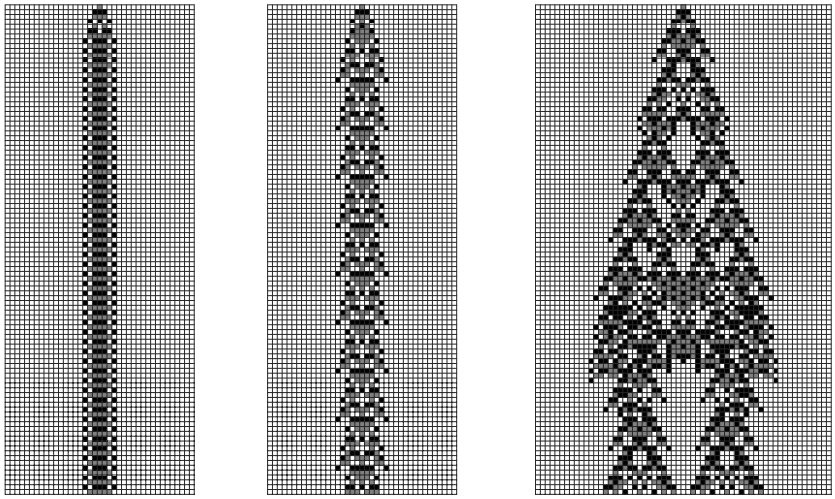
Most of the time the idea is to derive a mathematical formula that allows one to determine what the outcome of the evolution of the system will be without explicitly having to trace its steps.

And thus, for example, an early triumph of theoretical science was the derivation of a formula for the position of a single idealized planet orbiting a star. For given this formula one can just plug in numbers to work out where the planet will be at any point in the future, without ever explicitly having to trace the steps in its motion.

But part of what started my whole effort to develop the new kind of science in this book was the realization that there are many common systems for which no traditional mathematical formulas have ever been found that readily describe their overall behavior.

At first one might have thought this must be some kind of temporary issue, that could be overcome with sufficient cleverness. But from the discoveries in this book I have come to the conclusion that in fact it is not, and that instead it is one of the consequences of a very fundamental phenomenon that follows from the Principle of Computational Equivalence and that I call computational irreducibility.

If one views the evolution of a system as a computation, then each step in this evolution can be thought of as taking a certain amount of computational effort on the part of the system. But what traditional theoretical science in a sense implicitly relies on is that much of this effort is somehow unnecessary—and that in fact it should be possible to find the outcome of the evolution with much less effort.



Examples of computational reducibility and irreducibility in the evolution of cellular automata. The first two rules yield simple repetitive computationally reducible behavior in which the outcome after many steps can readily be deduced without tracing each step. The third rule yields behavior that appears to be computationally irreducible, so that its outcome can effectively be found only by explicitly tracing each step. The cellular automata shown here all have 3-color totalistic rules.

And certainly in the first two examples above this is the case. For just as with the orbit of an idealized planet there is in effect a straightforward formula that gives the state of each system after any

number of steps. So even though the systems themselves generate their behavior by going through a whole sequence of steps, we can readily shortcut this process and find the outcome with much less effort.

But what about the third example on the facing page? What does it take to find the outcome in this case? It is always possible to do an experiment and explicitly run the system for a certain number of steps and see how it behaves. But to have any kind of traditional theory one must find a shortcut that involves much less computation.

Yet from the picture on the facing page it is certainly not obvious how one might do this. And looking at the pictures on the next page it begins to seem quite implausible that there could ever in fact be any way to find a significant shortcut in the evolution of this system.

So while the behavior of the first two systems on the facing page is readily seen to be computationally reducible, the behavior of the third system appears instead to be computationally irreducible.

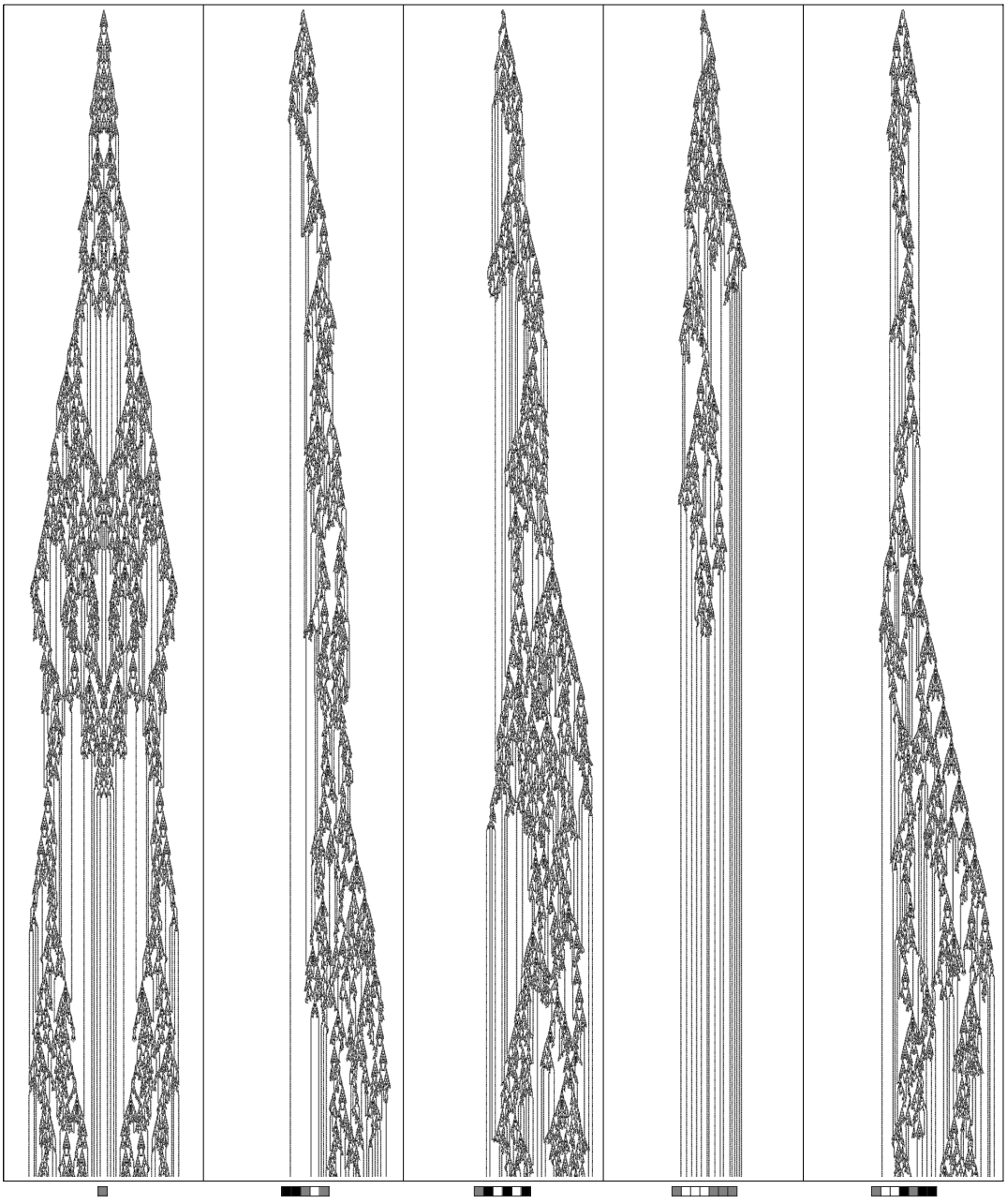
In traditional science it has usually been assumed that if one can succeed in finding definite underlying rules for a system then this means that ultimately there will always be a fairly easy way to predict how the system will behave.

Several decades ago chaos theory pointed out that to have enough information to make complete predictions one must in general know not only the rules for a system but also its complete initial conditions.

But now computational irreducibility leads to a much more fundamental problem with prediction. For it implies that even if in principle one has all the information one needs to work out how some particular system will behave, it can still take an irreducible amount of computational work actually to do this.

Indeed, whenever computational irreducibility exists in a system it means that in effect there can be no way to predict how the system will behave except by going through almost as many steps of computation as the evolution of the system itself.

In traditional science it has rarely even been recognized that there is a need to consider how systems that are used to make predictions actually operate. But what leads to the phenomenon of computational irreducibility is that there is in fact always a fundamental competition



5000 steps in the evolution of the third system from the previous page, starting from several initial conditions. The complexity of the behavior makes it seem inconceivable that there could ever be a procedure that would always immediately find its outcome.

between systems used to make predictions and systems whose behavior one tries to predict.

For if meaningful general predictions are to be possible, it must at some level be the case that the system making the predictions be able to outrun the system it is trying to predict. But for this to happen the system making the predictions must be able to perform more sophisticated computations than the system it is trying to predict.

In traditional science there has never seemed to be much problem with this. For it has normally been implicitly assumed that with our powers of mathematics and general thinking the computations we use to make predictions must be almost infinitely more sophisticated than those that occur in most systems in nature and elsewhere whose behavior we try to predict.

But the remarkable assertion that the Principle of Computational Equivalence makes is that this assumption is not correct, and that in fact almost any system whose behavior is not obviously simple performs computations that are in the end exactly equivalent in their sophistication.

So what this means is that systems one uses to make predictions cannot be expected to do computations that are any more sophisticated than the computations that occur in all sorts of systems whose behavior we might try to predict. And from this it follows that for many systems no systematic prediction can be done, so that there is no general way to shortcut their process of evolution, and as a result their behavior must be considered computationally irreducible.

If the behavior of a system is obviously simple—and is say either repetitive or nested—then it will always be computationally reducible. But it follows from the Principle of Computational Equivalence that in practically all other cases it will be computationally irreducible.

And this, I believe, is the fundamental reason that traditional theoretical science has never managed to get far in studying most types of systems whose behavior is not ultimately quite simple.

For the point is that at an underlying level this kind of science has always tried to rely on computational reducibility. And for example its whole idea of using mathematical formulas to describe behavior makes sense only when the behavior is computationally reducible.

So when computational irreducibility is present it is inevitable that the usual methods of traditional theoretical science will not work. And indeed I suspect the only reason that their failure has not been more obvious in the past is that theoretical science has typically tended to define its domain specifically in order to avoid phenomena that do not happen to be simple enough to be computationally reducible.

But one of the major features of the new kind of science that I have developed is that it does not have to make any such restriction. And indeed many of the systems that I study in this book are no doubt computationally irreducible. And that is why—unlike most traditional works of theoretical science—this book has very few mathematical formulas but a great many explicit pictures of the evolution of systems.

It has in the past couple of decades become increasingly common in practice to study systems by doing explicit computer simulations of their behavior. But normally it has been assumed that such simulations are ultimately just a convenient way to do what could otherwise be done with mathematical formulas.

But what my discoveries about computational irreducibility now imply is that this is not in fact the case, and that instead there are many common systems whose behavior cannot in the end be determined at all except by something like an explicit simulation.

Knowing that universal systems exist already tells one that this must be true at least in some situations. For consider trying to outrun the evolution of a universal system. Since such a system can emulate any system, it can in particular emulate any system that is trying to outrun it. And from this it follows that nothing can systematically outrun the universal system. For any system that could would in effect also have to be able to outrun itself.

But before the discoveries in this book one might have thought that this could be of little practical relevance. For it was believed that except among specially constructed systems universality was rare. And it was also assumed that even when universality was present, very special initial conditions would be needed if one was ever going to perform computations at anything like the level of sophistication involved in most methods of prediction.

But the Principle of Computational Equivalence asserts that this is not the case, and that in fact almost any system whose behavior is not obviously simple will exhibit universality and will perform sophisticated computations even with typical simple initial conditions.

So the result is that computational irreducibility can in the end be expected to be common, so that it should indeed be effectively impossible to outrun the evolution of all sorts of systems.

One slightly subtle issue in thinking about computational irreducibility is that given absolutely any system one can always at least nominally imagine speeding up its evolution by setting up a rule that for example just executes several steps of evolution at once.

But insofar as such a rule is itself more complicated it may in the end achieve no real reduction in computational effort. And what is more important, it turns out that when there is true computational reducibility its effect is usually much more dramatic.

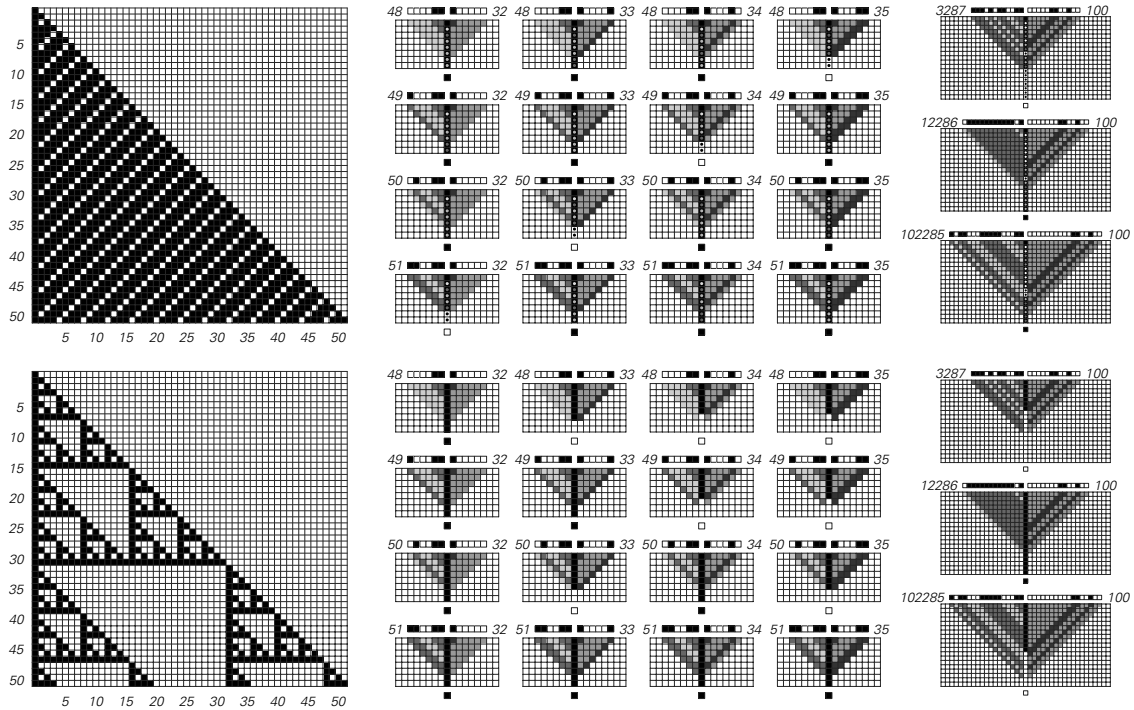
The pictures on the next page show typical examples based on cellular automata that exhibit repetitive and nested behavior. In the patterns on the left the color of each cell at any given step is in effect found by tracing the explicit evolution of the cellular automaton up to that step. But in the pictures on the right the results for particular cells are instead found by procedures that take much less computational effort.

These procedures are again based on cellular automata. But now what the cellular automata do is to take specifications of positions of cells, and then in effect compute directly from these the colors of cells.

The way things are set up the initial conditions for these cellular automata consist of digit sequences of numbers that give positions. The color of a particular cell is then found by evolving for a number of steps equal to the length of these input digit sequences.

And this means for example that the outcome of a million steps of evolution for either of the cellular automata on the left is now determined by just 20 steps of evolution, where 20 is the length of the base 2 digit sequence of the number 1,000,000.

And this turns out to be quite similar to what happens with typical mathematical formulas in traditional theoretical science. For the point of such formulas is usually to allow one to give a number as



Examples of computational reducibility in action. The pictures on the left show patterns produced by the ordinary evolution of cellular automata with elementary rules 188 and 60. The pictures on the right show how colors of particular cells in these patterns can be found with much less computational effort. In each case the position of a cell is specified by a pair of numbers given as base 2 digit sequences in the initial conditions for a cellular automaton. The evolution of the cellular automaton then quickly determines what the color of the cell at that position in the pattern on the left will be. For rule 188 the cellular automaton that does this involves 12 colors; for rule 60 it involves 6. In general, to find the color of a cell after t steps of rule 188 or rule 60 evolution takes about *Log[2, t]* steps. Compare page 608.

input, and then to compute directly something that corresponds, say, to the outcome of that number of steps in the evolution of a system.

In traditional mathematics it is normally assumed that once one has an explicit formula involving standard mathematical functions then one can in effect always evaluate this formula immediately.

But evaluating a formula—like anything else—is a computational process. And unless some digits effectively never matter, this process cannot normally take less steps than there are digits in its input.

Indeed, it could in principle be that the process could take a number of steps proportional to the numerical value of its input. But if this were so, then it would mean that evaluating the formula would

require as much effort as just tracing each step in the original process whose outcome the formula was supposed to give.

And the crucial point that turns out to be the basis for much of the success of traditional theoretical science is that in fact most standard mathematical functions can be evaluated in a number of steps that is far smaller than the numerical value of their input, and that instead normally grows only slowly with the length of the digit sequence of their input.

So the result of this is that if there is a traditional mathematical formula for the outcome of a process then almost always this means that the process must show great computational reducibility.

In practice, however, the vast majority of cases for which traditional mathematical formulas are known involve behavior that is ultimately either uniform or repetitive. And indeed, as we saw in Chapter 10, if one uses just standard mathematical functions then it is rather difficult even to reproduce many simple examples of nesting.

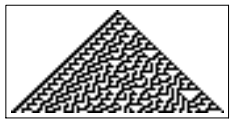
But as the pictures on the facing page and in Chapter 10 illustrate, if one allows more general kinds of underlying rules then it becomes quite straightforward to set up procedures that with very little computational effort can find the color of any element in any nested pattern.

So what about more complex patterns, like the rule 30 cellular automaton pattern at the bottom of the page?

When I first generated such patterns I spent a huge amount of time trying to analyze them and trying to find a procedure that would allow me to compute directly the color of each cell. And indeed it was the fact that I was never able to make much progress in doing this that first led me to consider the possibility that there could be a phenomenon like computational irreducibility.

And now, what the Principle of Computational Equivalence implies is that in fact almost any system whose behavior is not obviously simple will tend to exhibit computational irreducibility.

But particularly when the underlying rules are simple there is often still some superficial computational reducibility. And so, for example, in the rule 30 pattern on the right one can tell whether a cell at a given position has any chance of not being white just by doing a



An example of a pattern where it is difficult to compute directly the color of a particular cell.

very short computation that tests whether that position lies outside the center triangular region of the pattern. And in a class 4 cellular automaton such as rule 110 one can readily shortcut the process of evolution for at least a limited number of steps in places where there happen to be only a few well-separated localized structures present.

And indeed in general almost any regularities that we manage to recognize in the behavior of a system will tend to reflect some kind of computational reducibility in this behavior.

If one views the pattern of behavior as a piece of data, then as we discussed in Chapter 10 regularities in it allow a compressed description to be found. But the existence of a compressed description does not on its own imply computational reducibility. For any system that has simple rules and simple initial conditions—including for example rule 30—will always have such a description.

But what makes there be computational reducibility is when only a short computation is needed to find from the compressed description any feature of the actual behavior.

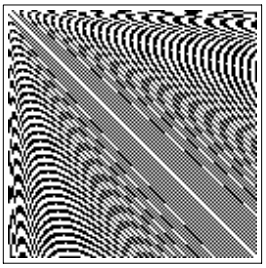
And it turns out that the kinds of compressed descriptions that can be obtained by the methods of perception and analysis that we use in practice and that we discussed in Chapter 10 all essentially have this property. So this is why regularities that we recognize by these methods do indeed reflect the presence of computational reducibility.

But as we saw in Chapter 10, in almost any case where there is not just repetitive or nested behavior, our normal powers of perception and analysis recognize very few regularities—even though at some level the behavior we see may still be generated by extremely simple rules.

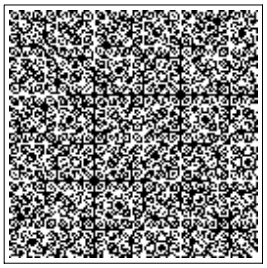
And this supports the assertion that beyond perhaps some small superficial amount of computational reducibility a great many systems are in the end computationally irreducible. And indeed this assertion explains, at least in part, why our methods of perception and analysis cannot be expected to go further in recognizing regularities.

But if behavior that we see looks complex to us, does this necessarily mean that it can exhibit no computational reducibility? One way to try to get an idea about this is just to construct patterns

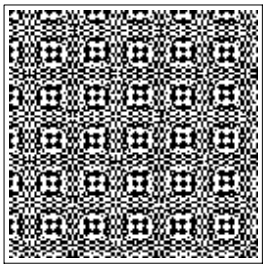
(a) If[Mod[Log[x y], 1] > 1/2, 1, 0]



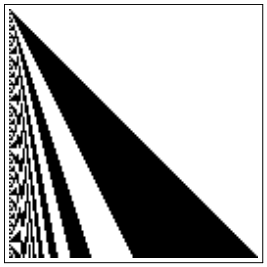
(b) If[Mod[Sqrt[x y], 1] > 1/2, 1, 0]



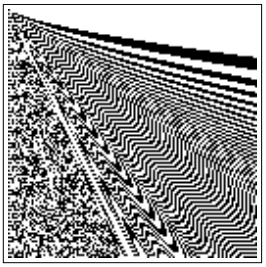
(c) If[Mod[Sin[x y], 1] > 1/2, 1, 0]



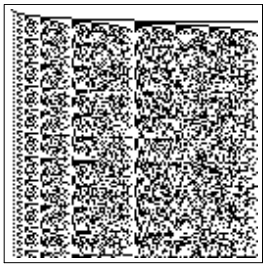
(d) If[Mod[Sin[x] + Sin[y], 1] > 1/2, 1, 0]



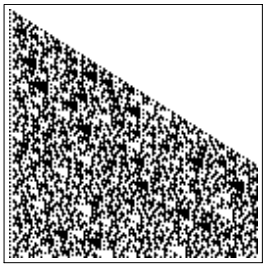
(e) Mod[Quotient[y, x], 2]



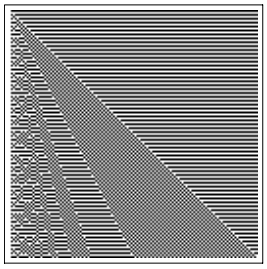
(f) Mod[Quotient[y^3, x^2], 2]



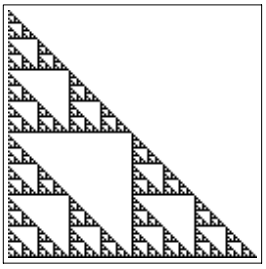
(g) Mod[Quotient[2^y, x], 2]



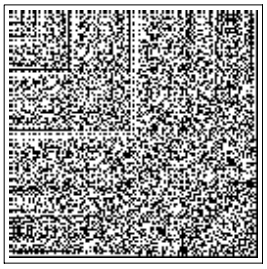
(h) Mod[Quotient[3^y, 2^x], 2]



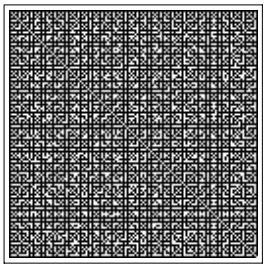
(i) Mod[Mod[y, x], 2]



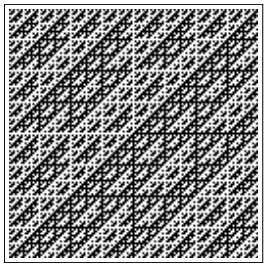
(j) Mod[Binomial[y, x], 2]



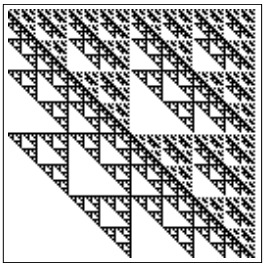
(k) Mod[DigitCount[x y, 2, 1], 2]



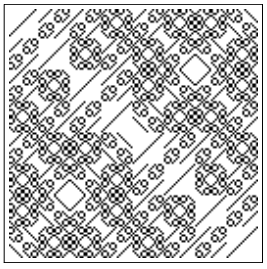
(l) If[GCD[x, y] == 1, 1, 0]



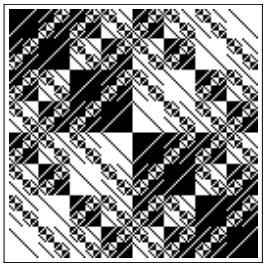
(m) Mod[d[x].d[y], 2]



(n) If[Count[d[y] - d[x], 1] == 1, 1, 0]



(o) If[Count[d[y] - d[x], 0] == 3, 1, 0]



(p) If[Count[d[y] - d[x], 0] > 3, 1, 0]

Examples of patterns set up so that a short computation can be used to determine the color of each cell from the numbers representing its position. Most such patterns look to us quite simple, but the examples shown here were specifically chosen to be ones that look more complicated. In most of them fairly standard mathematical functions are used, but in unusual combinations. In every picture both x and y run from 1 to 127. *d[n]* stands for *IntegerDigits[n, 2, 7]*. (h) is equivalent to digit sequences of powers of 3 in base 2 (see page 120). (j) is essentially Pascal's triangle (see page 611). (l) was discussed on page 613. (m) is a nested pattern seen on page 583. The only pattern that is known to be obtainable by evolving down the page according to a simple local rule is (j), which corresponds to the rule 60 elementary cellular automaton.

where we explicitly set up the color of each cell to be determined by some short computation from the numbers that represent its position.

When we look at such patterns most of them appear to us quite simple. But as the pictures on the previous page demonstrate, it turns out to be possible to find examples where this is not so, and where instead the patterns appear to us at least somewhat complex.

But for such patterns to yield meaningful examples of computational reducibility it must also be possible to produce them by some process of evolution—say by repeated application of a cellular automaton rule. Yet for the majority of cases shown here there is at least no obvious way to do this.

I have however found one class of systems—already mentioned in Chapter 10—whose behavior does not appear simple, but nevertheless turns out to be computationally reducible, as in the pictures on the facing page. However, I strongly suspect that systems like this are very rare, and that in the vast majority of cases where the behavior that we see in nature and elsewhere appears to us complex it is in the end indeed associated with computational irreducibility.

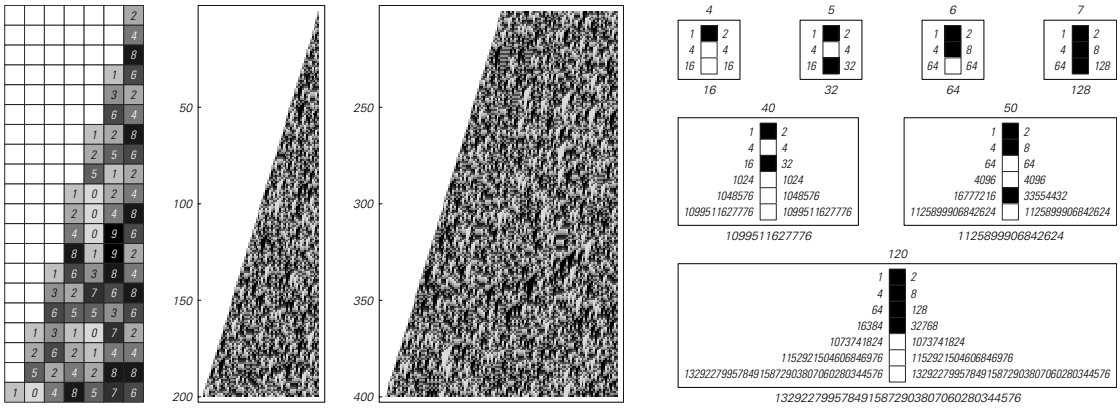
So what does this mean for science?

In the past it has normally been assumed that there is no ultimate limit on what science can be expected to do. And certainly the progress of science in recent centuries has been so impressive that it has become common to think that eventually it should yield an easy theory—perhaps a mathematical formula—for almost anything.

But the discovery of computational irreducibility now implies that this can fundamentally never happen, and that in fact there can be no easy theory for almost any behavior that seems to us complex.

It is not that one cannot find underlying rules for such behavior. Indeed, as I have argued in this book, particularly when they are formulated in terms of programs I suspect that such rules are often extremely simple. But the point is that to deduce the consequences of these rules can require irreducible amounts of computational effort.

One can always in effect do an experiment, and just watch the actual behavior of whatever system one wants to study. But what one



A system whose behavior looks complex but still turns out to be computationally reducible. The system is a cellular automaton with 10 possible colors for each cell. But it can also be viewed as a system based on numbers, in which successive rows are the base 10 digit sequences of successive powers of 2. And it turns out that there is a fast way to compute row n just from the base 2 digit sequence of n, as the pictures on the right illustrate. This procedure is based on the standard repeated squaring method of finding 2^n by starting from 2, and then successively squaring the numbers one gets, multiplying by 2 if the corresponding base 2 digit in n is 1. Using this procedure one can certainly compute the color of any cell on row n by doing about $n\,Log[n]^3$ operations—instead of the n^2 needed if one carried out the cellular automaton evolution explicitly.

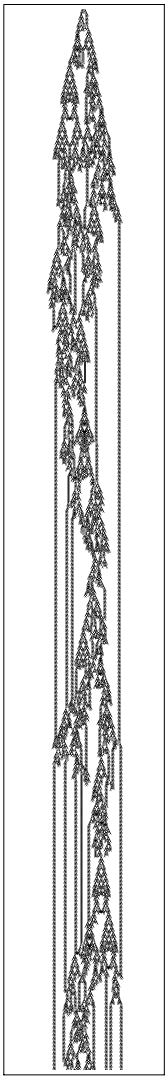
cannot in general do is to find an easy theory that will tell one without much effort what every aspect of this behavior will be.

So given this, can theoretical science still be useful at all?

The answer is definitely yes. For even in its most traditional form it can often deal quite well with those aspects of behavior that happen to be simple enough to be computationally reducible. And since one can never know in advance how far computational reducibility will go in a particular system it is always worthwhile at least to try applying the traditional methods of theoretical science.

But ultimately if computational irreducibility is present then these methods will fail. Yet there are still often many reasons to want to use abstract theoretical models rather than just doing experiments on actual systems in nature and elsewhere. And as the results in this book suggest, by using the right kinds of models much can be achieved.

Any accurate model for a system that exhibits computational irreducibility must at some level inevitably involve computations that are as sophisticated as those in the system itself. But as I have shown in



A cellular automaton whose behavior seems to show an analog of free will. Even though its underlying laws are definite—and simple—the behavior is complicated enough that many aspects of it seem to follow no definite laws. (The rule used is the same as on page 740.)

this book even systems with very simple underlying rules can still perform computations that are as sophisticated as in any system.

And what this means is that to capture the essential features even of systems with very complex behavior it can be sufficient to use models that have an extremely simple basic structure. Given these models the only way to find out what they do will usually be just to run them. But the point is that if the structure of the models is simple enough, and fits in well enough with what can be implemented efficiently on a practical computer, then it will often still be perfectly possible to find out many consequences of the model.

And that, in a sense, is what much of this book has been about.

The Phenomenon of Free Will

Ever since antiquity it has been a great mystery how the universe can follow definite laws while we as humans still often manage to make decisions about how to act in ways that seem quite free of obvious laws.

But from the discoveries in this book it finally now seems possible to give an explanation for this. And the key, I believe, is the phenomenon of computational irreducibility.

For what this phenomenon implies is that even though a system may follow definite underlying laws its overall behavior can still have aspects that fundamentally cannot be described by reasonable laws.

For if the evolution of a system corresponds to an irreducible computation then this means that the only way to work out how the system will behave is essentially to perform this computation—with the result that there can fundamentally be no laws that allow one to work out the behavior more directly.

And it is this, I believe, that is the ultimate origin of the apparent freedom of human will. For even though all the components of our brains presumably follow definite laws, I strongly suspect that their overall behavior corresponds to an irreducible computation whose outcome can never in effect be found by reasonable laws.

And indeed one can already see very much the same kind of thing going on in a simple system like the cellular automaton on the left. For