EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

Notes

General Notes

■ **Website.** A large amount of additional material related to this book and these notes will progressively be made available through the website www.wolframscience.com. (See also the copyright page at the beginning of the book.)

■ **The role of these notes.** The material in these notes is intended to be complementary to the main text, and is not always self-contained on its own. It is thus important to read these notes in parallel with the sections of the main text to which they refer, since some necessary points may be made only in the main text. Captions to pictures in the main text also often contain details that are not repeated in these notes.

■ **Writing style.** This book was not easy to write, not least because it contains many complex intellectual arguments presented in plain language. And in order to make these arguments as easy to understand as possible, I have had to adopt some rhetorical devices. Perhaps most annoying to those with a copyediting orientation will be my predilection for starting sentences with conjunctions. The main reason I have done this is to break up what would otherwise be extremely long sentences. For the points that I make are often sufficiently complex to require quite long explanations. And to make what I have written more readable than, say, a typical classic work of philosophy, I have broken these explanations into several sentences, necessarily with conjunctions at the beginning of each. Also annoying to some will be my widespread use of short paragraphs. In the main text I normally follow the principle that any paragraph should communicate just one basic idea. And my hope is then that after reading each paragraph readers will pause a moment to absorb each idea before going on to the next one. (This book introduces the third major distinct style of writing that I have used in publications. The first I developed for scientific papers; the second for documents like *The Mathematica Book*.)

■ **Billions.** Following standard American usage, billion in this book means 10^9, trillion 10^{12}, and so on.

■ **Clarity and modesty.** There is a common style of understated scientific writing to which I was once a devoted subscriber. But at some point I discovered that more significant results are usually incomprehensible if presented in this style. For unless one has a realistic understanding of how important something is, it is very difficult to place or absorb it. And so in writing this book I have chosen to explain straightforwardly the importance I believe my various results have. Perhaps I might avoid some criticism by a greater display of modesty, but the cost would be a drastic reduction in clarity.

■ **Explaining ideas.** In presenting major new ideas in a book such as this, there is a trade-off between trying to explain these ideas directly on their own, and using previous ideas to provide a context. For some readers there is a clear short-term benefit in referring to previous ideas, and in discussing to what extent they are right and wrong. But for other readers this approach is likely just to introduce confusion. And over the course of time the ideas that typical readers know will tend to shift. So to make this book as broadly accessible as possible what I mostly do is in the main text to discuss ideas as directly as I can—but then in these notes to outline their historical context. Occasionally in the main text I do mention existing ideas—though I try hard to avoid fads that I expect will not be widely remembered within a few years. Throughout the book my main goal is to explain new ideas, not to criticize ones from the past. Sometimes clarity demands that I say explicitly that something from the past is wrong, but generally I try to avoid this, preferring instead just to state whatever I now believe is true. No doubt this book will draw the ire of some of those with whose ideas its results do not agree, but much as I might like to do so, I cannot realistically avoid this just by the way I present what I have discovered.

■ **Technology references.** In an effort to make the main text of this book as timeless as possible, I have generally avoided

referring to everyday systems whose character or name I expect will change as technology advances. Inevitably, however, I do discuss computers, even though I fully expect that some of the terms and concepts I use in connection with them will end up seeming dated in a matter of a few decades.

▪ **Whimsy.** Cellular automata and most of the other systems in this book readily admit various kinds of whimsical descriptions. The rule 30 cellular automaton, for example, can be described as follows. Imagine a stadium full of people, with each person having two cards: one black and one white. Make the person in the middle of the top row of seats hold up a black card, and make everyone else in that row hold up a white card. Now each successive person in each successive row determines the color of the card they hold up by looking at the person directly above them, and above them immediately to their left and right, and then applying the simple rule on page 27. A photograph of the stadium will then show the pattern produced by rule 30. Descriptions like this may make abstract systems seem more connected to at least artificial everyday situations, but if the goal is to focus on fundamental ideas, as in this book, then such whimsy is, in my experience, normally just a major distraction.

▪ **Timeline of writing.** I worked on the writing of this book with few breaks for a little over ten years, beginning in June 1991, and ending in January 2002. The chapters were written roughly as follows: Chapter 1: 1991, 1999, 2001; Chapter 2: 1991–2; Chapter 3: 1992; Chapter 4: 1992–3; Chapter 5: 1993; Chapter 6: 1992–3; Chapter 7: 1994–6; Chapter 8: 1994–5, 1997; Chapter 9: 1995–8, 2001; Chapter 10: 1998–9; Chapter 11: 1995; Chapter 12: 1999–2001. Some sections of chapters (usually later ones) were added well after the rest. These notes were also sometimes written well after the main text of a given chapter.

▪ **Identifying new material.** The vast majority of results in this book have never appeared in published form before. A few were however included—implicitly or explicitly—in publications of mine from the early 1980s (see page 881). Whenever I am aware of antecedents to major material in the main text I have indicated this in the notes. Within the notes themselves, results that are given without historical discussion and without statements such as "it is known that" are generally new to this book. Researchers seeking further information should consult the website for the book.

▪ **Citations and references.** In developing the ideas described in this book I have looked at many thousands of books, papers and websites—and have interacted with hundreds of people (see page xiii). But rather than trying to give a huge list of specific references, I have instead included in these notes historical information tracing key contributions. From the names of concepts and people that I mention, it is straightforward to do web or database searches that give a vastly more complete picture of available references than could possibly fit in a book of manageable size—or than could be created correctly without immense scholarship. Note that while most current works of science tend to refer mainly just to very recent material, this book often refers to material that is centuries or even millennia old—in some ways more in the tradition of fields like philosophy.

▪ **Historical notes.** I have included extensive historical notes in this book in part out of respect for what has gone before, in part to provide context for ideas (and to see how current beliefs came to be as they are) and in part because the steps one goes through in understanding things may track steps that were gone through historically. Often in the book my conclusions in a particular field differ in a fundamental way from what has been traditional, and it has been important to me in confirming my understanding to study history and see how the conclusions I have reached were missed before. My discussion of science in this book is generally quite precise, being based among other things on computer experiments that can readily be reproduced. But my discussion of history is inevitably less precise. And while I have gone to considerable effort to ensure that its main elements are correct, ultimate objective confirmation is usually impossible. I have always tried to read original writings—for I have often found that later characterizations drop elements crucial for my purposes, or recast history to simplify pedagogy. But even for pieces of history where the people involved are still alive there are often no primary written records, leaving me to rely on secondary sources and recollections extracted in personal interviews—which are inevitably colored by later ideas and understanding. And while with sufficient effort it is usually possible to give fairly simple explanations for fundamental ideas in science, the same may not be true of their history. Looking at the historical notes in this book one striking feature is how often individuals of significant fame are mentioned—but not for the reason they are usually famous. And perhaps the explanation for this is in part that most of those who one can now see made contributions to the kinds of foundational issues I address were capable enough to have been successful at something—but without the whole context of this book they tended to view the types of results I discuss largely as curiosities, and so never tried to do much with them. Note that in mentioning people in connection with ideas and results, I have tried to concentrate on those who seemed to make the most essential contributions for my

purposes—even when this does not entirely agree with traditions or criteria in particular academic fields.

▪ **Dates.** Rather than following the usual academic practice of giving years when the discoveries were first published in books or journals, I have when possible given years when discoveries were first made. Note that I use a form like 1880s to refer to a decade, and 1800s to refer to a whole century.

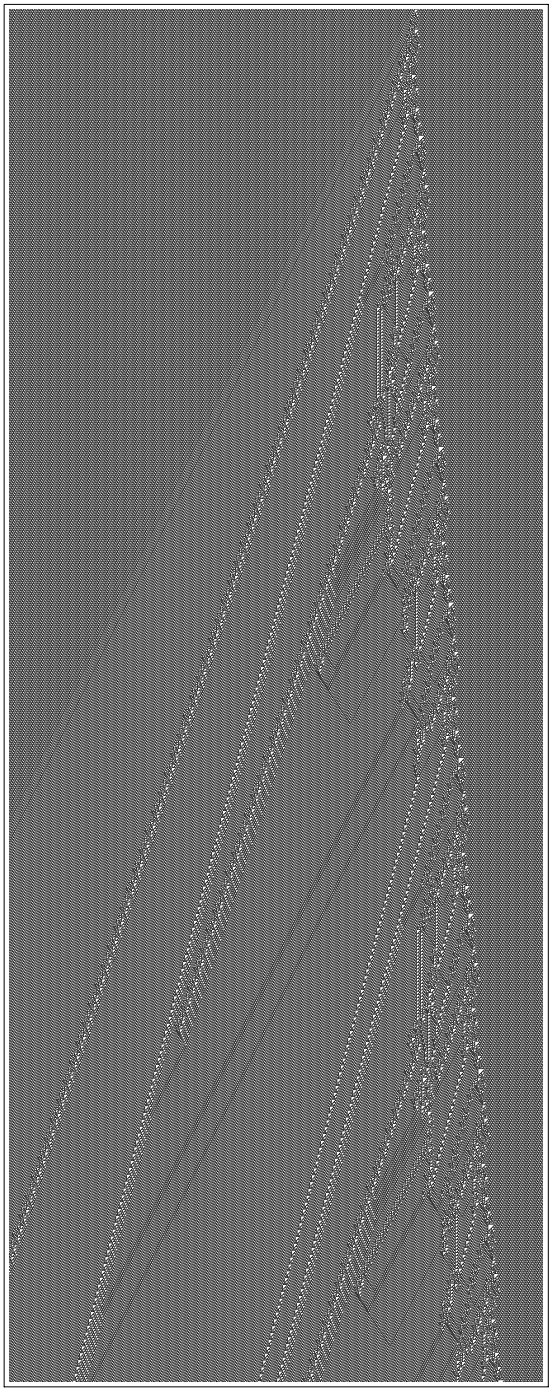
▪ **Autobiographical elements.** Every discovery in this book has some kind of specific personal story associated with it. Sometimes the story is quite straightforward; sometimes it is convoluted and colorful. But much as I enjoy recounting such stories, I have chosen not to make them part of this book.

▪ **Cover image.** The image on the cover of this book is derived from the first 440 or so steps (with perhaps 10 at each end cut off by trimming) of the pattern generated by evolution according to the rule 110 cellular automaton discussed on page 32, with an initial condition consisting of repeats of □□■■■■■ followed by repeats of ■■■■■□□. The picture on the right shows 3000 steps in this evolution. The central region grows by 1 cell every 2 steps on the left and 22 cells every 340 steps on the right. Many persistent structures emanate from the right-hand edge of the region. After just 29 steps, this edge takes on a form that repeats every 1700 steps. During each such cycle, a total of 65 persistent structures are produced, of 11 of the 15 kinds from page 292, and their interactions make the full repetition period 6800 steps.

▪ **Endpapers.** The goldenrod pages inside the front cover show the center 900 or so cells of the first 500 or so steps in the evolution of the rule 30 cellular automaton of page 29 from a single black cell. The pages inside the back cover show the next 500 or so steps.

▪ **Using color.** Aside from practicalities of printing, what made me decide not to use color in this book were issues of visual perception. For much as it is easier to read text in black and white, so also it is easier to assimilate detailed pictures if they are just in black and white. And in fact many types of images in this book show quite misleading features in color. In human visual perception the color of something tends to seem different depending on what is around it—so that for example a red element tends to look purple or pink if the elements around it are respectively blue or white. And particularly if there are few colors arranged in ways that are not visually familiar it is typical for this effect to make all sorts of spurious patterns appear.

▪ **Pictures in the book.** All the diagrammatic pictures in this book were created using *Mathematica*. (The photographs were also laid out and image-processed using *Mathematica*.) The ability of *Mathematica* to manipulate graphics in a symbolic



way was crucial—and was what ultimately made it possible for the book to have so many elaborate pictures.

To those familiar with book layout it may seem surprising that I was able to include so many pictures of so many different shapes and sizes without having to resort to a device like figure numbers. And indeed it required solving innumerable small geometrical puzzles to do so. But what ultimately made it possible was that the *Mathematica* programs for generating the pictures were almost always general enough that it was straightforward for me to get, say, a picture with a different number of cells or steps.

▪ **Hyphenation.** An unusual feature of the text in this book is that it almost never uses hyphenation; from seeing so much word-wrapped text on computers I at least have come to view hyphenation as an ugly and misleading device.

▪ **Book production system.** Beyond its actual content, the production of this book was a highly complex process that relied on the methodology for software releases developed at my company over the past fifteen years. Had I been starting the book now I would likely have authored all of it directly in *Mathematica* and *Publicon*. But a decade ago I made the decision to compose all the original source for the book in FrameMaker. This source was then processed by an elaborate automated procedure much like a standard software build. The first step involved converting a MIF version of the complete source into a *Mathematica* symbolic expression. Then within *Mathematica* various transformations and tests were done on this expression—with for example every program in these notes being formatted and broken into lines using rules similar to *Mathematica* StandardForm. The resulting symbolic expressions were then converted back to MIF, formatted in FrameMaker, and automatically output as PDF. (Note that special characters in programs are rendered using the new Mathematica-Sans font specifically created for the book.) (See also the colophon at the very end of the book.)

▪ **Printing.** Many of the pictures in this book have a rather different character from things that are normally printed. For unlike traditional diagrams consisting of separate visible elements—or photographs involving smooth gradations of color—they often for example contain hundreds of cells per inch, each in effect independently black or white. And to capture this properly required careful sheet-fed printing on paper smooth enough to avoid significant spreading of ink. (See also the colophon at the very end of the book.)

▪ **Index.** In the index to this book I have tried to cater both to those who have already read the book in detail, and to those who have not. My approach has generally been to include any term that might realistically come to mind when thinking of a given topic—or remembering what the book says about that topic. And this means that even if the book mentions a term only in passing, I have tended to include it if for one reason or another I think it is likely to be memorable to people with certain experience or interests. Note that looking up *Mathematica* functions used in connection with some issue is often a good way to identify related issues. In the actual building of the index in this book, sorting, processing and checking were done using a variety of automated *Mathematica* procedures, operating on a symbolic representation of the full text and index of the book. Often it is possible by reading an index to identify the important issues in a book. And to some extent that is possible here, though often the presence of more subentries just reflects material being more spread out, not more important.

▪ **People in the index.** Conventions for personal names vary considerably with culture and historical period. I have tried in the index to give all names in the form they might be used on standardized documents in the modern U.S. I have done standard transliterations from non-Latin character sets. I give in full those forenames that I believe are or were most commonly used by a particular individual; for other forenames (including for example Russian patronymics) I give only initials. I normally give formal versions of forenames—though for individuals I have personally known I give in the text the form of forenames I would normally use in addressing them. I have dropped all honorifics or titles, except when they significantly alter a name. When there are several versions of a name, I normally use the one that was current closest to the time of work I mention. For each person in the index I list the country or countries where that person predominantly worked. Note that this may not reflect where the person was born, educated, did military service, or died. Rather, it tries to indicate where the person did the majority of their work, particularly as it relates to this book. I generally refer to countries or regions by the names of their closest present-day approximations, as these might appear in postal addresses. When borders have changed, I tend to favor the country whose language is what the person normally speaks or spoke. I usually list countries in the order that a person has worked in them, ignoring repeats. Note that while many of the people listed are well known, extensive research (often through personal contacts, as well as institutional and government records) was required to track down quite a few of them. (Ending dates are obviously not included for people who died after the writing of this book was finished in January 2002.)

▪ **Notation.** In the main text, I have almost entirely avoided any kind of formal symbolic notation—usually relying

instead on diagrammatic pictures. In these notes, however, it will often be convenient to use such notation to give precise and compact representations of objects and operations. In the past, essentially the only large-scale notation available for theoretical science has been traditional mathematical notation. But on its own this would do me little good—for I need to represent not only traditional mathematics, but also more general rules and programs, as well as procedures and algorithms. But one of the reasons I created the *Mathematica* language was precisely to provide a much more general notation. So in these notes I use this language throughout as my notation. And this has many important advantages—and indeed it is hard to imagine that I would ever have been able to write these notes without it. One point is that it is completely uniform and standardized: there can never be any hidden assumptions or ambiguity about what a particular piece of notation means, since ultimately it is defined by the actual *Mathematica* software system and its documentation (see below). In cases where there is traditional mathematical notation for something, the corresponding *Mathematica* notation is normally almost identical—though occasionally a few details are changed to avoid ambiguity. The concept that everything is a symbolic expression allows *Mathematica* notation, however, to represent essentially any kind of abstract object. And when it comes to procedures and algorithms, the primitives in the *Mathematica* language are chosen to make typical steps easy to represent—with the result that a single line of *Mathematica* can often capture what would otherwise require many paragraphs of English text (and large amounts of pseudocode, or lower-level computer language code). Another very important practical feature of *Mathematica* notation is that by now a large number of people are familiar with it—certainly more than are for example familiar with sophisticated traditional notation in, say, mathematical logic. And the final and very critical advantage of *Mathematica* notation is that one can not only read it, but also actually execute it on a computer, and interact with it. And this makes it both vastly easier to apply and build on, and also easier to analyze and understand.

▪ ***Mathematica.*** I created *Mathematica* to be an integrated language and environment for computing in general, and technical computing in particular. Following its release in 1988, *Mathematica* has become very widely used in science, technology, education and elsewhere. (It is now also increasingly used as a component inside other software systems.)

Mathematica is available from Wolfram Research for all standard computer systems; much more information about it can be found on the web, especially from www.wolfram.com. There are many books about *Mathematica*—the original one being my *The Mathematica Book*.

The core of *Mathematica* is its language—which is based on the concept of symbolic programming. This language supports most traditional programming paradigms, but considerably generalizes them with the ideas of symbolic programming that I developed for it. In recent years there has started to be increasing use of the language component of *Mathematica* for all sorts of applications outside the area of technical computing where *Mathematica* as a whole has traditionally been most widely used.

The programs in these notes were created for *Mathematica* 4.1 (released 2000). They should run without any change in all subsequent versions of *Mathematica*, and the majority will also run in prior versions, all the way back to *Mathematica* 1 (released 1988) or *Mathematica* 2 (released 1990). Most of the programs require only the language component of *Mathematica*—and not its mathematical knowledge base—and so should run in all software systems powered by *Mathematica*, in which language capabilities are enabled.

Here are examples of how some of the basic *Mathematica* constructs used in the notes in this book work:

- Iteration

```
Nest[f, x, 3]⟶f[f[f[x]]]
NestList[f, x, 3]⟶{x, f[x], f[f[x]], f[f[f[x]]]}
Fold[f, x, {1, 2}]⟶f[f[x, 1], 2]
FoldList[f, x, {1, 2}]⟶{x, f[x, 1], f[f[x, 1], 2]}
```

- Functional operations

```
Function[x, x + k][a]⟶a + k
(# + k &)[a]⟶a + k
(r[#1] + s[#2] &)[a, b]⟶r[a] + s[b]
Map[f, {a, b, c}]⟶{f[a], f[b], f[c]}
Apply[f, {a, b, c}]⟶f[a, b, c]
Select[{1, 2, 3, 4, 5}, EvenQ]⟶{2, 4}
MapIndexed[f, {a, b, c}]⟶{f[a, {1}], f[b, {2}], f[c, {3}]}
```

- List manipulation

```
{a, b, c, d}[[3]]⟶c
{a, b, c, d}[[{2, 4, 3, 2}]]⟶{b, d, c, b}
Take[{a, b, c, d, e}, 2]⟶{a, b}
Drop[{a, b, c, d, e}, -2]⟶{a, b, c}
Rest[{a, b, c, d}]⟶{b, c, d}
ReplacePart[{a, b, c, d}, x, 3]⟶{a, b, x, d}
Length[{a, b, c}]⟶3
Range[5]⟶{1, 2, 3, 4, 5}
Table[f[i], {i, 4}]⟶{f[1], f[2], f[3], f[4]}
```

```
Table[f[i, j], {i, 2}, {j, 3}]⟶
  {{f[1, 1], f[1, 2], f[1, 3]}, {f[2, 1], f[2, 2], f[2, 3]}}
Array[f, {2, 2}]⟶{{f[1, 1], f[1, 2]}, {f[2, 1], f[2, 2]}}
Flatten[{{a, b}, {c}, {d, e}}]⟶{a, b, c, d, e}
Flatten[{{a, {b, c}}, {{d}, e}}, 1]⟶{a, {b, c}, {d}, e}
Partition[{a, b, c, d}, 2, 1]⟶{{a, b}, {b, c}, {c, d}}
Split[{a, a, a, b, b, a, a}]⟶{{a, a, a}, {b, b}, {a, a}}
ListConvolve[{a, b}, {1, 2, 3, 4, 5}]⟶
  {2 a + b, 3 a + 2 b, 4 a + 3 b, 5 a + 4 b}
Position[{a, b, c, a, a}, a]⟶{{1}, {4}, {5}}
RotateLeft[{a, b, c, d, e}, 2]⟶{c, d, e, a, b}
Join[{a, b, c}, {d, b}]⟶{a, b, c, d, b}
Union[{a, a, c, b, b}]⟶{a, b, c}
```

▪ Transformation rules

```
{a, b, c, d} /. b → p⟶{a, p, c, d}
{f[a], f[b], f[c]} /. f[a] → p⟶{p, f[b], f[c]}
{f[a], f[b], f[c]} /. f[x_] → p[x]⟶{p[a], p[b], p[c]}
{f[1], f[b], f[2]} /. f[x_Integer] → p[x]⟶{p[1], f[b], p[2]}
{f[1, 2], f[3], f[4, 5]} /. f[x_, y_] → x + y⟶{3, f[3], 9}
{f[1], g[2], f[2], g[3]} /. f[1] | g[_] → p⟶{p, p, f[2], p}
```

▪ Numerical functions

```
Quotient[207, 10]⟶20
Mod[207, 10]⟶7
Floor[1.45]⟶1
Ceiling[1.45]⟶2
IntegerDigits[13, 2]⟶{1, 1, 0, 1}
  IntegerDigits[13, 2, 6]⟶{0, 0, 1, 1, 0, 1}
DigitCount[13, 2, 1]⟶3
FromDigits[{1, 1, 0, 1}, 2]⟶13
```

The *Mathematica* programs in these notes are formatted in *Mathematica StandardForm*. The following table specifies how to enter these programs in *Mathematica InputForm*, using only ordinary keyboard characters:

π	Pi	∞	Infinity	e	E	i	I
x°	x Degree	x^y	x ^ y	$\sqrt{x}$	Sqrt[x]	$x \to y$	x -> y
$x \neq y$	x != y	$x \leq y$	x <= y	∂_y	D[y, x]	$\neg x$	!x
$x \wedge y$	x && y	$x \vee y$	x || y	$x \veebar y$	Xor[x, y]	$x \barwedge y$	Nand[x, y]

▪ **About the programs.** Like other aspects of the exposition in this book, I have gone to considerable effort to make the programs in these notes as clear and concise as possible. And I believe the final programs will be useful both to execute, and to read and study—if necessary without a computer. Most of the programs involve only built-in *Mathematica* functions, and so can be run in *Mathematica* without setting up any further definitions. (Many programs nevertheless contain variables that need to be assigned their values before the programs are run—as can be done for example with *Block[{k = 2}, program]*. When subsidiary functions are used, these functions also typically need to be defined before the programs are run—even though in these notes I often show the necessary definitions after the programs. Note that most of the programs do not explicitly do input checking or error generation. Only occasionally do the programs significantly sacrifice efficiency for elegance.) A good first step in understanding any program is to run it on a few inputs. The symbolic character of the *Mathematica* language also allows programs to be taken apart, so that their pieces can be run and analyzed separately. Careful study of the various programs in these notes should provide good background not only for implementing what I discuss in the book, but also for doing high-level programming of any kind. Many of the programs use several of the programming paradigms available in *Mathematica*—making it essentially impossible to capture their essence in any lower-level language. Note that a given program can essentially always be written in *Mathematica* in many different ways—though often other ways end up being vastly longer than the ones presented here. Material about the programs should be available at the book website—including for example some of the automated tests run to check the programs, as well as annotations about how the programs work.

▪ **Computer experiments.** Essentially all the computer experiments for this book were done using *Mathematica* running on a standard workstation-class computer, and later PC (initially on a 33 MHz NeXTstation, then on a 100 MHz HP 700 running NeXTSTEP, then on a 200 MHz P6 PC running Windows 95, and finally on 450 MHz, 700 MHz and faster PCs running Windows 95, and later Windows NT—with a Linux fileserver). For some larger searches earlier in the project, I wrote special-purpose C programs connected to *Mathematica* via *MathLink*. (Increasing computer speed and greater efficiency in successive versions of *Mathematica* have gradually almost eliminated my use of C.) In some cases I have run programs for many days or weeks, sometimes distributed via *MathLink* across a few hundred computers in my company's network. So far in my life the primary computer hardware systems I have used have been: Elliott 903 (1973–6); IBM 370 (1976–8); CDC 7600 (1978–9); VAX 11/780 (1980–2); Sun-1, 2, Ridge 32 (1982–4); CM-1 (1985); Sun-3 (1985–8); SPARC (1988–91); NeXT (1991–4); HP 700 (1995–6); PC (1996–). The primary languages have been: assembler (1973–6); FORTRAN (1976–9); C (1979–~1994); SMP (1980–6); *Mathematica* (1987–). (See also page 899.)

▪ **Educational issues.** The new kind of science in this book represents a unique educational opportunity. For it touches an immense range of important and compelling everyday phenomena and issues in science, yet to understand its key ideas requires no prior scientific or technical education. So this means that it is potentially realistic to use as the basis for an overall introduction to the ideas of science. And indeed having understood its basic elements, it becomes vastly easier to understand many aspects of traditional science, and to see how they fit into the whole framework of knowledge.

No doubt there will at first be a tendency to follow the progression of scientific history and to present the ideas of this book only at an advanced stage in the educational process, after teaching many aspects of traditional science. But it is fairly clear that it is vastly easier to explain much of what is in this book than to explain many ideas in traditional science. For among other things the new kind of science in this book does not rely on elaborate abstract concepts from traditional mathematics; instead it is based mostly just on pictures, and on ideas that have become increasingly familiar from practical use of computers. And in fact, in my experience, with good presentation, surprisingly young children are able to grasp many key ideas in this book—even if their knowledge of mathematics does not go beyond the simplest operations on numbers.

Over the past fifty or so years traditional mathematics has become a core part of education. And while its more elementary aspects are certainly crucial for everyday modern life, beyond basic algebra its central place in education must presumably be justified more on the basis of promoting overall patterns of thinking than in supplying specific factual knowledge of everyday relevance. But in fact I believe that the basic aspects of the new kind of science in this book in many ways provide more suitable material for general education than traditional mathematics. They involve some of the same kinds of precise thinking, but do not rely on abstract concepts that are potentially very difficult to communicate. And insofar as they involve the development of technical expertise, it is in the direction of computing—which is vastly more relevant to modern life than advanced mathematics.

The new kind of science in this book connects in all sorts of ways with mathematics and the existing sciences—and it can be used at an educational level to place some of the fundamental ideas in these areas in a clearer context. In computer science it can also be used as a rich source of basic examples—much as physics is used as a source of basic examples in traditional mathematics education.

A remarkable feature of the new kind of science in this book is that it makes genuine research accessible to people with almost no specific technical knowledge. For it is almost certain that experiments on, say, some specific cellular automaton whose rule has been picked at random from a large set will never have been done before. To conclude anything interesting from such experiments nevertheless requires certain scientific methodology and judgement—but from an educational point of view this represents a uniquely accessible environment in which to develop such skills.

In many fields, advanced education seems useful only if one intends to pursue those specific fields. But a few fields such as physics are notable for being sources of individuals with broadly applicable skills. I believe that the new kind of science in this book will in time serve a similar role.

▪ **Reading this book.** This is a long book densely packed with ideas and results, and to read all of it carefully is a major undertaking. The first section of Chapter 1 provides a basic—though compressed—overview of some of key ideas. Chapter 2 describes some of the basic results that led me to develop the new kind of science in the book. Every subsequent chapter in one way or another builds on earlier ones. Some people will probably find the sweeping conclusions of the final chapter of the book the most interesting; others will probably be more interested in specific results and applications in earlier chapters.

These notes are never necessary for the basic flow of any of the arguments I make in the book—though they often provide context and important supporting information, as well as considerable amounts of new primary material. Specialists in particular fields should be sure to read the notes that relate to their fields before they draw any final conclusions about what I have to say.

I have written this book with considerable care, and I believe that to those seriously interested in its contents, it will repay careful and repeated reading. Note that in the main text I have tried to emphasize important points by various kinds of stylistic devices. But in packing as much as possible into these notes I have often been unable to do this. And in general these notes have a high enough information density that it will be rare that everything they say can readily be assimilated in just one reading, even if it is quite careful.

▪ **Learning the new kind of science.** There will, I hope, be many who want to learn about what is in this book, whether out of general intellectual interest, to apply it in some way or another, or to participate in its further development. But regardless of the purpose, the best first step will certainly be to read as much of this book as possible with care. In time

there will doubtless also be all sorts of additional material and educational options available. But ultimately the key to a real understanding is to experience ideas for oneself. And for the new kind of science in this book this is in a sense unprecedentedly easy, for all it requires is a standard computer on which to do computer experiments.

At first the best thing is probably just to repeat some of the experiments I describe in this book—using the software and resources described at the website, or perhaps just by typing in some of the programs in these notes. And even if one can already see the result of an experiment in a picture in this book, it has been my consistent observation that one internalizes results of experiments much better if one gets them by running a program oneself than if one just sees them printed in a book. To get a deeper understanding, however, one invariably needs to try formulating experiments for oneself. One might wonder, for example, what would happen if some particular system were run for more steps than I show in this book. Would the system go on doing what one sees in the book, or might it start doing something quite different? With the appropriate setup, one can immediately run a program to find out. Often one will have some kind of guess about what the answer should be. At first—if my own experience is a guide—this guess will quite often be wrong. But gradually, after seeing what happens in enough cases, one will begin to develop a correct and robust new intuition. Realistically this seems to take several months even for the most talented and open-minded people. But as the new intuition matures, ideas in this book like the Principle of Computational Equivalence, that may at first seem hard to believe, will slowly come to seem almost obvious.

For someone to assimilate all of the new kind of science I describe in this book will take a very significant time. Indeed, in a traditional educational setting I expect that it will require an investment of years comparable to learning an area like physics. How long it will take a given individual to get to the point of being able to do something specific with the new kind of science in this book will depend greatly on their background and particular goals. But in almost any case a crucial practical step—if it has not already been taken—will be to learn well *Mathematica* and the language it embodies. For although most simple programs can be implemented in almost any computational environment, not using the capabilities of *Mathematica* will be an immediate handicap—which, for example, would certainly have prevented me from discovering the vast majority of what is now in this book.

▪ **Developing the new kind of science.** Up to this point in its history the science in this book has essentially been just my personal project. But now that the book is out, all sorts of other people can begin to participate—adding their own personal achievements to the development of the intellectual structure that I have built in this book.

The first obvious but crucial thing to do is to explain and interpret what is already in the book. For although this is a long book that I have tried to write as clearly as possible, there is immensely more that can and should be said—in many different ways—about almost all the ideas and results it contains. Sometimes a more technical presentation may be useful; sometimes a less technical one. Sometimes it will be helpful to make more connections to some existing area of thought or scholarship. And sometimes particular ideas and results in this book will just benefit from the emphasis of having a whole paper or book or website devoted to them.

One of my goals in this book has been to answer the most obvious questions about each of the subjects I address. And at this I believe I have been moderately successful. But the science I have developed in this book opens up an area so vast that the twenty years I have spent investigating it have allowed me to explore only tiny parts. And indeed from almost every page of this book there are all sorts of new questions that emerge. In fact, even about systems that I have studied as extensively as cellular automata I am always amazed at just how easy it is to identify worthwhile questions that have not yet been addressed. And in general the ideas and methods of this book seem to yield an unending stream of important questions of a remarkable range of different kinds.

On the website associated with this book I plan to maintain a list of questions that I believe are of particular interest. The questions will be of many kinds and at many levels. Some it will be possible to address just by fairly straightforward but organized computer experimentation, while others will benefit from varying levels of technical skill and knowledge from existing areas of science, mathematics or elsewhere.

Like any serious intellectual pursuit, doing well the new kind of science in this book is not easy. In writing the book I have put great effort into explaining things in straightforward ways. But the fact that in some particular case I may have succeeded does not mean that the underlying science was easy. And in fact my uniform experience has been that to make progress in the kind of science I describe in this book requires at a raw intellectual level at least as much as any traditional area of science. The kind of extensive detailed technical knowledge that characterizes most traditional areas of science is usually not needed—though it can be helpful. But if anything, greater clarity and organization of thought is

needed than in areas where there is existing technical formalism to fall back on. At a practical level the most important basic skill is probably *Mathematica* programming. For it is crucial to be able to try out new ideas and experiments quickly—and in my experience it is also important to have the discipline of formulating things in the precise language of *Mathematica*.

One feature of this book is that it covers a broad area and comes to very broad conclusions. But to get to the point of being able to do this has taken me twenty years of gradually building up from specific detailed results and ideas. And I have no doubt that in the future essentially all significant contributions will also be made by building on foundations of specific detailed facts. And indeed, what I expect to be the mainstay of the science that develops from this book is the gradual accumulation of more and more knowledge of a variety of detailed concrete kinds.

I have tried in this book to lead by example in defining the way I believe things should be done. Probably the single most important principle that I have followed is just to try to keep everything as simple as possible. Study the simplest systems. Ask the most obvious questions. Search for the most straightforward explanations. For among other things, this is ultimately how the most useful and powerful results are obtained. Not that it is easy to do this. For while in the end it may be possible to get to something simple and elegant, it often takes huge intellectual effort to see just how this can be done. And without great tenacity there is a tremendous tendency to stop before one has gone far enough.

In most existing fields of science there are so many technicalities to learn and keep current on that it is rare for anyone but a professional scientist to be able to make any significant contribution. But in the new kind of science that I describe in this book I believe that at least at first there will be opportunities for a much broader range of people to make contributions. In existing fields of science their largely closed communities tend to maintain standards of quality mostly through direct institutional and personal contact. Yet particularly when there are technical aspects to a field it is also comparatively easy for practitioners to assess a piece of work just from the overall way it handles and presents its technicalities. And in fact there are obvious analogs of this in the new kind of science that I describe in this book. First, there is the issue of whether tools like computers are used in effective ways. But in many ways more central is whether there is a certain basic level of clarity and simplicity to a piece of work. Often it is difficult to achieve this. But the point is that the skills necessary to do so correspond rather directly to the ones necessary to carry out the actual science itself well.

▪ **Applications.** At the core of this book is a body of ideas and results that define a new kind of basic science. And I have no doubt that in time this will yield a remarkably broad range of applications. And sometimes—particularly in technology—these applications may be quite straightforward and direct. But if the objective is to develop a model for some specific system in nature or elsewhere it is almost inevitable that this will not be easy. For while I believe that the basic science that I develop in this book provides a remarkably powerful new framework, coming up with an actual model requires all sorts of detailed work and analysis. Certainly it would be wonderful if one could just take the ideas and results in this book and somehow immediately use them to create models for all sorts of systems. And indeed—particularly from the examples I give in Chapter 8—there will probably be at least a few cases where this can be done. But most of the time nothing like it will be possible. And instead—just as in any other framework—there will be no choice but first to learn all sorts of details of a system, and then to use judgement and creativity to see which of them are really essential to a model and which are not. (See also page 364.)

The Foundations for a New Kind of Science

An Outline of Basic Ideas

▪ **Mathematics in science.** The main event usually viewed as marking the beginning of the modern mathematical approach to science was the publication of Isaac Newton's 1687 book *Mathematical Principles of Natural Philosophy* (the *Principia*). The idea that mathematics might be relevant to science nevertheless had long precursors in both practical and philosophical traditions. Before 500 BC the Babylonians were using arithmetic to describe and predict astronomical data. And by 500 BC the Pythagoreans had come to believe that all natural phenomena should somehow be reducible to relationships between numbers. Many Greek philosophers then discussed the general concept that nature should be amenable to abstract reasoning of the kind used in mathematics. And at a more practical level, the results and methodology of Euclid's work on geometry from around 300 BC became the basis for studies in astronomy, optics and mechanics, notably by Archimedes and Ptolemy. In medieval times there were some doubts about the utility of mathematics in science, and in the late 1200s, for example, Albertus Magnus made the statement that "many of the geometer's figures are not found in natural bodies, and many natural figures, particularly those of animals and plants, are not determinable by the art of geometry". Roger Bacon nevertheless wrote in 1267 that "mathematics is the door and key to the sciences", and by the 1500s it was often believed that for science to be meaningful it must somehow follow the systematic character of mathematics. (Typical of the time was the statement of Leonardo da Vinci that "no human inquiry can be called science unless it pursues its path through mathematical exposition and demonstration".) Around the end of the 1500s Galileo began to develop more explicit connections between concepts in mathematics and in physics, and concluded that the universe could be understood only in the "language of mathematics", whose "characters are triangles, circles and other geometric figures". What Isaac Newton then did was in effect to suggest that natural systems are at some fundamental level actually governed by purely abstract laws that can be specified in terms of mathematical equations. This idea has met with its greatest success in physics, where for the past three centuries essentially every major theory has been formulated in terms of mathematical equations. Starting in the mid-1800s, it has also had increasing success in chemistry. And in the past century, it has had a few scattered successes in dealing with simpler phenomena in fields like biology and economics. But despite the vast range of phenomena in nature that have never successfully been described in mathematical terms, it has become quite universally assumed that, as David Hilbert put it in 1900, "mathematics is the foundation of all exact knowledge of natural phenomena". There continue to be theories in science that are not explicitly mathematical—examples being continental drift and evolution by natural selection—but, as for example Alfred Whitehead stated in 1911, it is generally believed that "all science as it grows toward perfection becomes mathematical in its ideas".

▪ **Definition of mathematics.** When I use the term "mathematics" in this book what I mean is that field of human endeavor that has in practice traditionally been called mathematics. One could in principle imagine defining mathematics to encompass all studies of abstract systems, and indeed this was in essence the definition that I had in mind when I chose the name *Mathematica*. But in practice mathematics has defined itself to be vastly narrower, and to include, for example, nothing like the majority of the programs that I discuss in this book. Indeed, in many respects, what is called mathematics today can be seen as a direct extension of the particular notions of arithmetic and geometry that apparently arose in Babylonian times. Typical dictionary definitions reflect this by describing mathematics as the study of number and space, together with their abstractions and generalizations. And even logic—an abstract system that dates from antiquity—is not normally

considered part of mainstream mathematics. Particularly over the past century the defining characteristic of research in mathematics has increasingly been the use of theorem and proof methodology. And while some generalization has occurred in the types of systems being studied, it has usually been much limited by the desire to maintain the validity of some set of theorems (see page 793). This emphasis on theorems has also led to a focus on equations that statically state facts rather than on rules that define actions, as in most of the systems in this book. But despite all these issues, many mathematicians implicitly tend to assume that somehow mathematics as it is practiced is universal, and that any possible abstract system will be covered by some area of mathematics or another. The results of this book, however, make it quite clear that this is not the case, and that in fact traditional mathematics has reached only a tiny fraction of all the kinds of abstract systems that can in principle be studied.

■ **Reasons for mathematics in science.** It is not surprising that there should be issues in science to which mathematics is relevant, since until about a century ago the whole purpose of mathematics was at some level thought of as being to provide abstract idealizations of aspects of physical reality (with the consequence that concepts like dimensions above 3 and transfinite numbers were not readily accepted as meaningful even in mathematics). But there is absolutely no reason to think that the specific concepts that have arisen so far in the history of mathematics should cover all of science, and indeed in this book I give extensive evidence that they do not. At times the role of mathematics in science has been used in philosophy as an indicator of the ultimate power of human thinking. In the mid-1900s, especially among physicists, there was occasionally some surprise expressed about the effectiveness of mathematics in the natural sciences. One explanation advanced by Albert Einstein was that the only physical laws we can recognize are ones that are easy to express in our system of mathematics.

■ **History of programs and nature.** Given the idea of using programs as a basis for describing nature, one can go back in history and find at least a few rough precursors of this idea. Around 100 AD, for example, following earlier Greek thinking, Lucretius made the somewhat vague suggestion that the universe might consist of atoms assembled according to grammatical rules like letters and words in human language. From the Pythagoreans around 500 BC through Ptolemy around 150 AD to the early work of Johannes Kepler around 1595 there was the notion that the planets might follow definite geometrical rules like the elements of a mechanical clock. But following the work of Isaac Newton in the late 1600s it increasingly came to be believed that systems could only meaningfully be described by the mathematical equations they satisfy, and not by any explicit mechanism or rules. The failure of the concept of ether and the rise of quantum mechanics in the early 1900s strengthened this view to the point where at least in physics mechanistic explanations of any kind became largely disreputable. (Starting in the 1800s systems based on very simple rules were nevertheless used in studies of genetics and heredity.) With the advent of electronics and computers in the 1940s and 1950s, models like neural networks and cellular automata began to be introduced, primarily in biology (see pages 876 and 1099). But in essentially all cases they were viewed just as approximations to models based on traditional mathematical equations. In the 1960s and 1970s there arose in the early computer hacker community the general idea that the universe might somehow operate like a program. But attempts to engineer explicit features of our universe using constructs from practical programming were unsuccessful, and the idea largely fell into disrepute (see page 1026). Nevertheless, starting in the 1970s many programs were written to simulate all sorts of scientific and technological systems, and often these programs in effect defined the models used. But in almost all cases the elements of the models were firmly based on traditional mathematical equations, and the programs themselves were highly complex, and not much like the simple programs I discuss in this book. (See also pages 363 and 992.)

■ **Extensions of mathematics.** See page 793.

■ **The role of logic.** In addition to standard mathematics, the formal system most widely discussed since antiquity is logic (see page 1099). And starting with Aristotle there was in fact a long tradition of trying to use logic as a framework for drawing conclusions about nature. In the early 1600s the experimental method was suggested as a better alternative. And after mathematics began to show extensive success in describing nature in the late 1600s no further large-scale efforts to do this on the basis of logic appear to have been made. It is conceivable that Gottfried Leibniz might have tried in the late 1600s, but when his work was followed up in the late 1800s by Gottlob Frege and others the emphasis was on building up mathematics, not natural science, from logic (see page 1149). And indeed by this point logic was viewed mostly as a possible representation of human thought—and not as a formal system relevant to nature. So when computers arose it was their numerical and mathematical rather than logical capabilities that were normally assumed relevant for natural science. But in the early 1980s the cellular automata that I studied I often characterized as being based on logical rules, rather than traditional mathematical ones. However, as

we will see on page 806, traditional logic is in fact in many ways very narrow compared to the whole range of rules based on simple programs that I actually consider in this book.

▪ **Complexity and theology.** Both complexity and order in the natural world have been cited as evidence for an intelligent creator (compare page 1195). Early mythologies most often assume that the universe started in chaos, with a supernatural being adding order, then creating a series of specific complex natural systems. In Greek philosophy it was commonly thought that the regularities seen in astronomy and elsewhere (such as the obvious circular shapes of the Sun and Moon) were reflections of perfect mathematical forms associated with divine beings. About complexity Aristotle did note that what nature makes is "finer than art", though this was not central to his arguments about causes of natural phenomena. By the beginning of the Christian era, however, there is evidence of a general belief that the complexity of nature must be the work of a supernatural being—and for example there are statements in the Bible that can be read in this way. Around 1270 Thomas Aquinas gave as an argument for the existence of God the fact that things in nature seem to "act for an end" (as revealed for example by always acting in the same way), and thus must have been specifically designed with that end in mind. In astronomy, as specific natural laws began to be discovered, the role of God began to recede somewhat, with Isaac Newton claiming, for example, that God must have first set the planets on their courses, but then mathematical laws took over to govern their subsequent behavior. Particularly in biology, however, the so-called "argument by design" became ever more popular. Typical was John Ray's 1691 book *The Wisdom of God Manifested in the Works of the Creation,* which gave a long series of examples from biology that it claimed were so complex that they must be the work of a supernatural being. By the early 1800s, such ideas had led to the field of natural theology, and William Paley gave the much quoted argument that if it took a sophisticated human watchmaker to construct a watch, then the only plausible explanation for the vastly greater complexity of biological systems was that they must have been created by a supernatural being. Following the publication of Charles Darwin's *Origin of Species* in 1859 many scientists began to argue that natural selection could explain all the basic phenomena of biology, and although some religious groups maintained strong resistance, it was widely assumed by the mid-1900s that no other explanation was needed. In fact, however, just how complexity arises was never really resolved, and in the end I believe that it is only with the ideas of this book that this can successfully be done.

▪ **Artifacts and natural systems.** See page 828.

▪ **Complexity and science.** Ever since antiquity science has tended to see its main purpose as being the study of regularities—and this has meant that insofar as complexity is viewed as an absence of regularities, it has tended to be ignored or avoided. There have however been occasional discussions of various general aspects of complexity and what can account for them. Thus, for example, by 200 BC the Epicureans were discussing the idea that varied and complex forms in nature could be made up from arrangements of small numbers of types of elementary atoms in much the same way as varied and complex written texts are made up from small numbers of types of letters. And although its consequences were remarkably confused, the notion of a single underlying substance that could be transmuted into anything—living or not—was also a centerpiece of alchemy. Starting in the 1600s successes in physics and discoveries like the circulation of blood led to the idea that it should be possible to explain the operation of almost any natural system in essentially mechanical terms—leading for example René Descartes to claim in 1637 that we should one day be able to explain the operation of a tree just like we do a clock. But as mathematical methods developed, they seemed to apply mainly to physical systems, and not for example to biological ones. And indeed Immanuel Kant wrote in 1790 that "it is absurd to hope that another Newton will arise in the future who will make comprehensible to us the production of a blade of grass according to natural laws". In the late 1700s and early 1800s mathematical methods began to be used in economics and later in studying populations. And partly influenced by results from this, Charles Darwin in 1859 suggested natural selection as the basis for many phenomena in biology, including complexity. By the late 1800s advances in chemistry had established that biological systems were made of the same basic components as physical ones. But biology still continued to concentrate on very specific observations—with no serious theoretical discussion of anything as general as the phenomenon of complexity. In the 1800s statistics was increasingly viewed as providing a scientific approach to complex processes in practical social systems. And in the late 1800s statistical mechanics was then used as a basis for analyzing complex microscopic processes in physics. Most of the advances in physics in the late 1800s and early 1900s in effect avoided complexity by concentrating on properties and systems simple enough to be described by explicit mathematical formulas. And when other fields tried in the early and mid-1900s to imitate successes in physics, they too generally tended to concentrate on issues that seemed amenable to explicit mathematical

formulas. Within mathematics itself—especially in number theory and the three-body problem—there were calculations that yielded results that seemed complex. But normally this complexity was viewed just as something to be overcome—either by looking at things in a different way, or by proving more powerful theorems—and not as something to be studied or even much commented on in its own right.

In the 1940s, however, successes in the analysis of logistical and electronic systems led to discussion of the idea that it might be possible to set up some sort of general approach to complex systems—especially biological and social ones. And by the late 1940s the cybernetics movement was becoming increasingly popular—with Norbert Wiener emphasizing feedback control and stochastic differential equations, and John von Neumann and others emphasizing systems based on networks of elements often modelled after neurons. There were spinoffs such as control theory and game theory, but little progress was made on core issues of complexity, and already by the mid-1950s what began to dominate were vague discussions involving fashionable issues in areas such as psychiatry and anthropology. There also emerged a tradition of robotics and artificial intelligence, and a few of the systems that were built or simulated did show some complexity of behavior (see page 879). But in most cases this was viewed just as something to be overcome in order to achieve the engineering objectives sought. Particularly in the 1960s there was discussion of complexity in large human organizations—especially in connection with the development of management science and the features of various forms of hierarchy—and there emerged what was called systems theory, which in practice typically involved simulating networks of differential equations, often representing relationships in flowcharts. Attempts were for example made at worldwide models, but by the 1970s their results—especially in economics—were being discredited. (Similar methods are nevertheless used today, especially in environmental modelling.)

With its strong emphasis on simple laws and measurements of numbers, physics has normally tended to define itself to avoid complexity. But from at least the 1940s, issues of complexity were nevertheless occasionally mentioned by physicists as important, most often in connection with fluid turbulence or features of nonlinear differential equations. Questions about pattern formation, particularly in biology and in relation to thermodynamics, led to a sequence of studies of reaction-diffusion equations, which by the 1970s were being presented as relevant to general issues of complexity, under names like self-organization, synergetics and dissipative structures. By the late 1970s the work of Benoit Mandelbrot on fractals provided an important example of a general approach to addressing a certain kind of complexity. And chaos theory—with its basis in the mathematics of dynamical systems theory—also began to become popular in the late 1970s, being discussed particularly in connection with fluid turbulence. In essentially all cases, however, the emphasis remained on trying to find some aspect of complex behavior that could be summarized by a single number or a traditional mathematical equation.

As discussed on pages 44–50, there were by the beginning of the 1980s various kinds of abstract systems whose rules were simple but which had nevertheless shown complex behavior, particularly in computer simulations. But usually this was considered largely a curiosity, and there was no particular sense that there might be a general phenomenon of complexity that could be of central interest, say in natural science. And indeed there remained an almost universal belief that to capture any complexity of real scientific relevance one must have a complex underlying model. My work on cellular automata in the early 1980s provided strong evidence, however, that complex behavior very much like what was seen in nature could in fact arise in a very general way from remarkably simple underlying rules. And starting around the mid-1980s it began to be not uncommon to hear the statement that complex behavior can arise from simple rules—though often there was great confusion about just what this was actually saying, and what, for example, should be considered complex behavior, or a simple rule.

That complexity could be identified as a coherent phenomenon that could be studied scientifically in its own right was something I began to emphasize around 1984. And having created the beginnings of what I considered to be the necessary intellectual structure, I started to try to develop an organizational structure to allow what I called complex systems research to spread. Some of what I did had fairly immediate effects, but much did not, and by late 1986 I had started building *Mathematica* and decided to pursue my own scientific interests in a more independent way (see page 20). By the late 1980s, however, there was widespread discussion of what was by then being called complexity theory. (I had avoided this name to prevent confusion with the largely unrelated field of computational complexity theory). And indeed many of the points I had made about the promise of the field were being enthusiastically repeated in popular accounts—and there were starting to be quite a number of new institutions devoted to the field. (A notable example was the Santa Fe Institute, whose orientation towards complexity seems to have been a quite direct consequence of my efforts.)

But despite all this, no major new scientific developments were forthcoming—not least because there was a tremendous tendency to ignore the idea of simple underlying rules and of what I had discovered in cellular automata, and instead to set up computer simulations with rules far too complicated to allow them to be used in studying fundamental questions. And combined with a predilection for considering issues in the social and biological sciences that seem hard to pin down, this led to considerable skepticism among many scientists—with the result that by the mid-1990s the field was to some extent in retreat—though the statement that complexity is somehow an important and fundamental issue has continued to be emphasized especially in studies of ecological and business systems.

Watching the history of the field of complexity theory has made it particularly clear to me that without a major new intellectual structure complexity cannot realistically be studied in a meaningful scientific way. But it is now just such a structure that I believe I have finally been able to set up in this book.

Relations to Other Areas

■ **Page 7 · Mathematics.** I discuss the implications of this book for the foundations of mathematics mainly on pages 772–821 and in the rather extensive corresponding notes. With a sufficiently general definition of mathematics, however, the whole core of the book can in fact be viewed as a work of experimental mathematics. And even with a more traditional definition, this is at least true of much of my discussion of systems based on numbers in Chapter 4. The notes to almost all chapters of the book contain a great many new mathematical results, mostly emerging from my analysis of some of the simpler behavior considered in the book. Pages 606–620 and 737–750 discuss in general the capabilities of mathematical analysis, while pages 588–597 address the foundations of statistics. Note that some ideas and results highly relevant to current frontiers in mathematics appear in some rather unexpected places in the book. Specific examples include the parameter space sets that I discuss in connection with shapes of plant leaves on page 407, and the minimal axioms for logic that I discuss on page 810. A more general example is the issue of smooth objects arising from combinatorial data that I discuss in Chapter 9 in connection with the nature of space in fundamental physics.

■ **Page 8 · Physics.** I discuss general mechanisms and models relevant for physical systems in Chapter 7, specific types of everyday physical systems in Chapter 8, and applications to basic foundational problems in physics in Chapter 9. I mention some further fundamental issues in physics around page 730 and in chemistry on page 1193.

■ **Page 8 · Biology.** The main place I discuss applications to biology is on pages 383–429 of Chapter 8, where I consider first general questions about biology and evolution, and then more specific issues about growth and pattern in biological organisms. I consider visual and auditory perception on pages 577–588, and the operation of brains on pages 620–631. I also discuss the definition of life on pages 823 and 1178, as well as mentioning protein folding and structure on pages 1003 and 1184.

■ **Page 9 · Social and related sciences.** I discuss the particular example of financial systems on pages 429–432, and make some general comments on page 1014. The end of Chapter 10, as well as some parts of Chapter 12, also discuss various issues that can be viewed as foundational questions.

■ **Page 10 · Computer science.** Chapter 11 as well as parts of Chapter 12 (especially pages 753–771) address foundational issues in computer science. Chapter 3 uses standard computer science models such as Turing machines and register machines as examples of simple programs. In many places in the book—especially these notes—I discuss all sorts of specific problems and issues of direct relevance to current computer science. Examples include cryptography (pages 598–606), Boolean functions (pages 616–619 and 806–814), user interfaces (page 1102) and quantum computing (page 1147).

■ **Page 10 · Philosophy.** Chapter 12 is the main place I address traditional philosophical issues. On pages 363–369 of Chapter 8, however, I discuss some general issues of modelling, and in Chapter 10 I consider at length not only practical but also foundational questions about perception and to some extent general thinking and consciousness. (See page 1196.)

■ **Page 11 · Technology.** The notes to this book mention many specific technological connections, and I expect that many of the models and methods of analysis that I use in the book can be applied quite directly for technological purposes. I discuss foundational questions about technology mainly on pages 840–843.

■ **Scope of existing sciences.** One might imagine that physics would for example concern itself with all aspects of physical systems, biology with all aspects of biological systems, and so on. But in fact as they are actually practiced most of the traditional sciences are much narrower in scope. Historically what has typically happened is that in each science a certain way of thinking has emerged as the most successful. And then over the course of time, the scope of the science itself has come to be defined to encompass just those issues that this

way of thinking is able to address. So when a new phenomenon is observed, a particular science will typically tend to focus on just those aspects of the phenomenon that can be studied by whatever way of thinking has been adopted in that science. And when the phenomenon involves substantial complexity, what has in the past usually happened is that simpler and simpler aspects are investigated until one is found that is simple enough to analyze using the chosen way of thinking.

The Personal Story of the Science in This Book

▪ **Page 17 · Statistical physics cover.** The pictures show disks representing idealized molecules bouncing around in a box, and the book claims that as time goes on there is almost inevitably increasing randomization. The pictures were made in about 1964 by Berni Alder and Frederick Reif from oscilloscope output from the LARC computer at what was then Lawrence Radiation Laboratory. A total of 40 disks were started with positions and velocities determined by a middle-square random number generator (see page 975), and their motion was followed for about 10 collision times—after which roundoff errors in the 64-bit numbers used had grown too big. From the point of view of this book the randomization seen in these pictures is in large part just a reflection of the fact that a random sequence of digits were used in the initial conditions. But what the discoveries in this book show is that such randomness can also be generated without any such random input—finally clarifying some very basic issues in statistical physics. (See page 441.)

▪ **Page 17 · My 1973 computer experiments.** I used a British Elliott 903 computer with 8 kilowords of 18-bit ferrite core memory. The assembly language program that I wrote filled up a fair fraction of the memory. The system that I looked at was a 2D cellular automaton with discrete particles colliding on a square grid. Had I not been concerned with physics-like conservation laws, or had I used something other than a square grid, the teleprinter output that I generated would have shown randomization. (See page 999.)

▪ **Page 19 · Computer printouts.** The printouts show a series of elementary cellular automata started from random initial conditions (see page 232). I generated them in 1981 using a C program running on a VAX 11/780 computer with an early version of the Unix operating system. (See also page 880.)

▪ **Timeline.** Major periods in my work have been:

- 1974–1980: particle physics and cosmology
- 1979–1981: developing SMP computer algebra system
- 1981–1986: cellular automata etc.
- 1986–1991: intensive *Mathematica* development
- 1991–2001: writing this book

(Wolfram Research, Inc. was founded in 1987; *Mathematica* 1.0 was released June 23, 1988; the company and successive versions of *Mathematica* continue to be major parts of my life.)

▪ **Detailed history.** See pages 880–882.

The Crucial Experiment

How Do Simple Programs Behave?

▪ **Implementing cellular automata.** It is convenient to represent the state of a cellular automaton at each step by a list such as *{0, 0, 1, 0, 0}*, where *0* corresponds to a white cell and *1* to a black cell. An initial condition consisting of *n* white cells with one black cell in the middle can then be obtained with the function (see below for comments on this and other *Mathematica* functions)

```
CenterList[n_Integer] :=
  ReplacePart[Table[0, {n}], 1, Ceiling[n/2]]
```

For cellular automata of the kind discussed in this chapter, the rule can also be represented by a list. Thus, for example, rule 30 on page 27 corresponds to the list *{0, 0, 0, 1, 1, 1, 1, 0}*. (The numbering of rules is discussed on page 53.) In general, the list for a particular rule can be obtained with the function

```
ElementaryRule[num_Integer] := IntegerDigits[num, 2, 8]
```

Given a rule together with a list representing the state *a* of a cellular automaton at a particular step, the following simple function gives the state at the next step:

```
CAStep[rule_List, a_List] :=
  rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]
```

A list of states corresponding to evolution for *t* steps can then be obtained with

```
CAEvolveList[rule_, init_List, t_Integer] :=
  NestList[CAStep[rule, #] &, init, t]
```

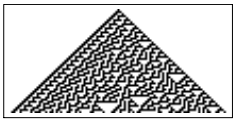
Graphics of this evolution can be generated using

```
CAGraphics[history_List] := Graphics[
    Raster[1 - Reverse[history]] AspectRatio → Automatic]
```

And having set up the definitions above, the *Mathematica* input

```
Show[CAGraphics[CAEvolveList[
      ElementaryRule[30], CenterList[103], 50]]]
```

will generate the image:



The description just given should be adequate for most cellular automaton simulations. In some earlier versions of *Mathematica* a considerably faster version of the program can be created by using the definition

```
CAStep = Compile[{{rule, _Integer, 1}, {a, _Integer, 1}},
    rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]]
```

In addition, in *Mathematica* 4 and above, one can use

```
CAStep[rule_, a_] := rule[[8 - ListConvolve[{1, 2, 4}, a, 2]]]
```

or directly in terms of the rule number *num*

```
Sign[BitAnd[2^ListConvolve[{1, 2, 4}, a, 2], num]]
```

(In versions of *Mathematica* subsequent to the release of this book the built-in *CellularAutomaton* function can be used, as discussed on page 867.) It is also possible to have *CAStep* call the following external C language program via *MathLink*—though typically with successive versions of *Mathematica* the speed advantage obtained will be progressively less significant:

```
#include "mathlink.h"

main(argc, argv)
int argc; char *argv[];
{
MLMain(argc, argv);
}

void casteps(revrule, rlen, a, n, steps)
int *revrule, rlen, *a, n, steps;
{
int i, *ap, t, tp;

for (i = 0; i <steps; i++)
   {
   a[0] = a[n-2]; /* right boundary */
   a[n-1] = a[1]; /* left boundary */

   t = a[0];
   for (ap = a+1; ap <= a+n-2; ap++)
      {
      tp = ap[0];
      ap[0] = revrule[ap[1]+2*(tp + 2*t)];
      t = tp;
      }
   }

MLPutIntegerList(stdlink, a, n);
}
```

The linkage of this external program to the *Mathematica* function *CAStep* is achieved with the following *MathLink* template (note the optional third argument which allows

CAStep to perform several steps of cellular automaton evolution at a time):

```
:Begin:
:Function: casteps
:Pattern: CAStep[rule_List, a_List, steps_Integer:1]
:Arguments: {Reverse[rule], a, steps}
:ArgumentTypes: {IntegerList, IntegerList, Integer}
:ReturnType: Manual
:End:
```

There are a couple of tricky issues in the C program above. First, cellular automaton rules are always defined to use the old values of neighbors in determining the new value of any particular cell. But since the C program explicitly updates values sequentially from left to right, the left-hand neighbor of a particular cell will already have been given its new value when one tries to updates the cell itself. As a result, it is necessary to store the old value of the left-hand neighbor in a temporary variable in order to make it available for updating the cell itself. (Another approach to this problem is to maintain two copies of the array of cells, and to interchange pointers to them after every step in the cellular automaton evolution.)

Another tricky point in cellular automaton programs concerns boundary conditions. Since in a practical computer one can use only a finite array of cells, one must decide how the cellular automaton rule is to be applied to the cells at each end of the array. In both the *Mathematica* and the C programs above, we effectively use a cyclic array, in which the left neighbor of the leftmost cell is taken to be rightmost cell, and vice versa. In the C program, this is implemented by explicitly copying the value of the leftmost cell to the rightmost position in the array, and vice versa, before updating the values in the array. (In a sense there is a bug in the program in that the update only puts new values into $n-2$ of the n array elements.)

■ **Comments on *Mathematica* functions.** *CenterList* works by first creating a list of n 0's, then replacing the middle 0 by a 1. (In *Mathematica* 4 and above *PadLeft[{1}, n, 0, Floor[n/2]]* can be used instead.) *ElementaryRule* works by converting *num* into a base 2 digit sequence, padding with zeros on the left so as to make a list of length 8. The scheme for numbering rules works so that if the value of a particular cell is q, the value of its left neighbor is p, and the value of its right neighbor is r, then the element at position $8-(r+2(q+2p))$ in the list obtained from *ElementaryRule* will give the new value of the cell.

CAStep uses the fact that *Mathematica* can manipulate all the elements in a list at once. *RotateLeft[a]* and *RotateRight[a]* make shifted versions of the original list of cell values a. Then when these lists are added together, their corresponding elements are combined, as in $\{p, q, r\}+\{s, t, u\} \rightarrow \{p+s, q+t, r+u\}$. The result is that a list is produced which specifies for each cell which element of the rule applies to that cell. The actual list of new cell values is then generated by using the fact that $\{i, j, k\}[[\{2, 1, 1, 3, 2\}]] \rightarrow \{j, i, i, k, j\}$. Note that by using *RotateLeft* and *RotateRight* one automatically gets cyclic boundary conditions.

CAEvolveList applies *CAStep* t times. Many other evolution functions in these notes use the same mechanism. In general *NestList[s[r, #] &, i, 2]* $\rightarrow$ *{i, s[r, i], s[r, s[r, i]]}*, etc.

■ **Bitwise optimizations.** The C program above stores each cell value in a separate element of an integer array. But since every value must be either 0 or 1, it can in fact be encoded by just a single bit. And since integer variables in practical computers typically involve 32 or 64 bits, the values of many cells can be packed into a single integer variable. The main point of this is that typical machine instructions operate in parallel on all the bits in such a variable. And thus for example the values of all cells represented by an integer variable a can be updated in parallel according to rule 30 by the single C statement

```
a = a>>1 ^ (a | a<<1);
```

This statement, however, will only update the specific block of cells encoded in a. Gluing together updates to a sequence of such blocks requires slightly intricate code. (It is much easier to implement in *Mathematica*—as discussed above—since there functions like *BitXor* can operate on integers of any length.) In general, bitwise optimizations require representing cellular automaton rules not by simple look-up tables but rather by Boolean expressions, which must be derived for each rule and can be quite complicated (see page 869). Applying the rules can however be made faster by using bitslicing to avoid shift operations. The idea is to store the cellular automaton configuration in, say, m variables *w[i]* whose bits correspond respectively to the cell values $\{a_1, a_{m+1}, a_{2m+1}, \ldots\}$, $\{a_2, a_{m+2}, a_{2m+2}, \ldots\}$, $\{a_3, \ldots\}$, etc. This then makes the left and right neighbors of the j^{th} bit in *w[i]* be the j^{th} bits in *w[i-1]* and *w[i+1]*—so that for example a step of rule 30 evolution can be achieved just by *w[i] = w[i-1]^(w[i] | w[i+1])* with no shift operations needed (except in boundary conditions on *w[0]* and *w[m-1]*). If many steps of evolution are required, it is sufficient just to pack all cell values at the beginning, and unpack them at the end.

■ **More general rules.** The programs given so far are for cellular automata with rules of the specific kind described in this chapter. In general, however, a 1D cellular automaton rule can be given as a set of explicit replacements for all

possible blocks of cells in each neighborhood (see page 60). Thus, for example, rule 30 can be given as

```
{{1, 1, 1} → 0, {1, 1, 0} → 0, {1, 0, 1} → 0, {1, 0, 0} → 1,
  {0, 1, 1} → 1, {0, 1, 0} → 1, {0, 0, 1} → 1, {0, 0, 0} → 0}
```

To use rules in this form, *CAStep* can be rewritten as

```
CAStep[rule_, a_List] :=
  Transpose[{RotateRight[a], a, RotateLeft[a]}] /. rule
```

or

```
CAStep[rule_, a_List] := Partition[a, 3, 1, 2] /. rule
```

The rules that are given can now contain patterns, so that rule 90, for example, can be written as

```
{{1, _, 1} → 0, {1, _, 0} → 1, {0, _, 1} → 1, {0, _, 0} → 0}
```

But how can one set up a program that can handle rules in several different forms? A convenient approach is to put a "wrapper" around each rule that specifies what form the rule is in. Then, for example, one can define

```
CAStep[ElementaryCARule[rule_List], a_List] :=
  rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]
CAStep[GeneralCARule[rule_, r_Integer : 1], a_List] :=
  Partition[a, 2 r + 1, 1, r + 1] /. rule
CAStep[FunctionCARule[f_, r_Integer : 1], a_List] :=
  Map[f, Partition[a, 2 r + 1, 1, r + 1]]
```

Note that the second two definitions have been generalized to allow rules that involve r neighbors on each side. In each case, the use of *Partition* could be replaced by *Transpose[Table[RotateLeft[a, i], {i, -r, r}]]*. For efficiency in early versions of *Mathematica*, explicit rule lists in the second definition can be preprocessed using *Dispatch[rules]*, and functions in the third definition preprocessed using *Compile[{{x, _Integer, 1}}, body]*.

I discuss the implementation of totalistic cellular automata on page 886, and of higher-dimensional cellular automata on page 927.

▪ **Built-in cellular automaton function.** Versions of *Mathematica* subsequent to the release of this book will include a very general function for cellular automaton evolution. The description is as follows (see also page 886):

CellularAutomaton[rnum, init, t] generates a list representing the evolution of cellular automaton rule *rnum* from initial condition *init* for t steps.

CellularAutomaton[rnum, init, t, {off$_1$, off$_2$ ...}] keeps only the parts of the evolution list with the specified offsets.

Possible settings for *rnum* are:

n	$k = 2$, $r = 1$, elementary rule
$\{n, k\}$	general nearest-neighbor rule with k colors
$\{n, k, r\}$	general rule with k colors and range r
$\{n, k, \{r_1, r_2, \ldots, r_d\}\}$	d-dimensional rule with $(2 r_1 + 1) \times (2 r_2 + 1) \times \ldots \times (2 r_d + 1)$ neighborhood
$\{n, k, \{\{off_1\}, \{off_2\}, \ldots, \{off_s\}\}\}$	rule with neighbors at specified offsets
$\{n, \{k, 1\}\}$	k-color nearest-neighbor totalistic rule
$\{n, \{k, 1\}, r\}$	k-color range r totalistic rule
$\{n, \{k, \{wt_1, wt_2, \ldots\}\}, rspec\}$	rule in which neighbor i is assigned weight wt_i
$\{fun, \{\}, rspec\}$	applies the function *fun* to each list of neighbors, with a second argument of the step number

▪ *CellularAutomaton[{n, k}, ...]* is equivalent to *CellularAutomaton[{n, {k, {k^2, k, 1}}}, ...]*. ▪ Common forms for 2D cellular automata include:

$\{n, \{k, 1\}, \{1, 1\}\}$	9-neighbor totalistic rule
$\{n, \{k, \{\{0, 1, 0\}, \{1, 1, 1\}, \{0, 1, 0\}\}, \{1, 1\}\}\}$	5-neighbor totalistic rule
$\{n, \{k, \{\{0, k, 0\}, \{k, 1, k\}, \{0, k, 0\}\}, \{1, 1\}\}\}$	5-neighbor outer totalistic rule
$\{n + k^5 (k - 1), \{k, \{\{0, 1, 0\}, \{1, 4 k + 1, 1\}, \{0, 1, 0\}\}, \{1, 1\}\}\}$	5-neighbor growth rule

▪ Normally, all elements in *init* and the evolution list are integers between 0 and k-1. ▪ But when a general function is used, the elements of *init* and the evolution list do not have to be integers. ▪ The second argument passed to *fun* is the step number, starting at 0. ▪ Initial conditions are constructed from *init* as follows:

$\{a_1, a_2, \ldots\}$	explicit list of values a_i, assumed cyclic
$\{\{a_1, a_2, \ldots\}, b\}$	values a_i superimposed on a b background
$\{\{a_1, a_2, \ldots\}, \{b_1, b_2, \ldots\}\}$	values a_i superimposed on a background of repetitions of $b_1, b_2, \ldots$
$\{\{\{a_{11}, a_{12}, \ldots\}, off_1\}, \{a_{21}, \ldots\}, off_2\}, \ldots\}, bspec\}$	values a_{ij} at offsets off_i on a background
$\{\{a_{11}, a_{12}, \ldots\}, \{a_{21}, \ldots\}, \ldots\}$	explicit list of values in two dimensions
$\{aspec, bspec\}$	values in d dimensions with d-dimensional padding

▪ The first element of *aspec* is superimposed on the background at the first position in the positive direction in each coordinate relative to the origin. This means that *bspec[[1,1,...]]* is aligned with *aspec[[1, 1, ...]]*. ▪ Time offsets off_t are specified as follows:

All	all steps 0 through t (default)
u	steps 0 through u
-1	last step (step t)
$\{u\}$	step u
$\{u_1, u_2\}$	steps u_1 through u_2
$\{u_1, u_2, du\}$	steps $u_1, u_1 + du, \ldots$

▪ *CellularAutomaton[rnum, init, t]* generates an evolution list of length t+1. ▪ The initial condition is taken to have offset 0. ▪ Space offsets off_x are specified as follows:

All	all cells that can be affected by the specified initial condition
Automatic	all cells in the region that differs from the background
0	cell aligned with beginning of *aspec*
x	cells at offsets up to x on the right
$-x$	cells at offsets up to x on the left
$\{x\}$	cell at offset x to the right
$\{-x\}$	cell at offset x to the left
$\{x_1, x_2\}$	cells at offsets x_1 through x_2
$\{x_1, x_2, dx\}$	cells $x_1, x_1 + dx, \ldots$

▪ In one dimension, the first element of *aspec* is taken by default to have space offset 0. ▪ In any number of dimensions, *aspec[[1, 1, ...]]* is taken by default to have space offset *{0, 0, ...}*. ▪ Each element of the evolution list produced by *CellularAutomaton* is always the same size. ▪ With an initial condition specified by an *aspec* of width w, the region that can be affected after t steps by a cellular automaton with a

rule of range r has width $w + 2\,r\,t$. ▪ If no *bspec* background is specified, space offsets of *All* and *Automatic* will include every cell in *aspec*. ▪ A space offset of *All* includes all cells that can be affected by the initial condition. ▪ A space offset of *Automatic* can be used to trim off background from the sides of a cellular automaton pattern. ▪ In working out how wide a region to keep, *Automatic* only looks at results on steps specified by off_t.

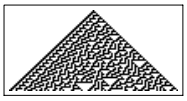
Some examples include:

This gives the array of values obtained by running rule 30 for 3 steps, starting from an initial condition consisting of a single 1 surrounded by 0's.

```
In[1] := CellularAutomaton[30, {{1}, 0}, 3]
Out[1]= {{0, 0, 0, 1, 0, 0, 0}, {0, 0, 1, 1, 1, 0, 0}, {0, 1, 1, 0, 0, 1, 0}, {1, 1, 0, 1, 1, 1, 1}}
```

This runs rule 30 for 50 steps and makes a picture of the result.

```
In[2] := Show[RasterGraphics[CellularAutomaton[30, {{1}, 0}, 50]]]
```

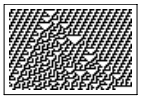


If all values in the initial condition are given explicitly, they are in effect assumed to continue cyclically. The runs rule 30 with 5 cells for 3 steps.

```
In[3] := CellularAutomaton[30, {1, 0, 0, 1, 0}, 3]
Out[3]= {{1, 0, 0, 1, 0}, {1, 1, 1, 1, 0}, {1, 0, 0, 0, 0}, {1, 1, 0, 0, 1}}
```

This starts from {1,1} on an infinite background of repeating {1,0,1,1} blocks. By default, only the region of the pattern affected by the {1,1} is given.

```
In[4] := Show[RasterGraphics[CellularAutomaton[30, {{1, 1}, {1, 0, 1, 1}}, 50]]]
```

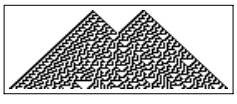


This gives all cells that could possibly be affected, whether or not they are.

```
In[5] := Show[RasterGraphics[CellularAutomaton[30,
                {{1, 1}, {1, 0, 1, 1}}, 50, {All, All}]]]
```

This places blocks in the initial conditions at offsets -10 and 20.

```
In[6] := Show[RasterGraphics[CellularAutomaton[30,
                {{{1}, {-10}}, {{1, 1}, {20}}}, 0}, 50]]]
```



This gives only the last row after running for 10 steps.

```
In[7] := CellularAutomaton[30, {{1}, 0}, 10, -1]
Out[7]= {{1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0}}
```

This runs for 5 steps, giving the cells on the 3 center columns at each step.

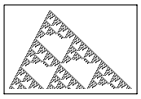
```
In[8] := CellularAutomaton[30, {{1}, 0}, 5, {All, {-1, 1}}]
Out[8]= {{0, 1, 0}, {1, 1, 1}, {1, 0, 0}, {0, 1, 1}, {0, 1, 0}, {1, 1, 1}}
```

This picks out every other cell in space and time, starting 200 cells to the left.

```
In[9] := Show[RasterGraphics[CellularAutomaton[30, {{1}, 0}, 100,
                {{1, 100, 2}, {-200, 200, 2}}]]]
```

This runs the general $k=3$, $r=1$ rule with rule number 921408.

```
In[10] := Show[RasterGraphics[CellularAutomaton[{921408, 3, 1}, {{1}, 0}, 100]]]
```

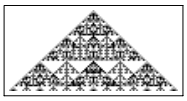


This runs the totalistic $k=3$, $r=1$ rule with code 867.

```
In[11] := Show[RasterGraphics[CellularAutomaton[{867, {3, 1}, 1}, {{1}, 0}, 50]]]
```

This uses a rule based on applying a function to each neighborhood of cells.

```
In[12] := Show[RasterGraphics[CellularAutomaton[
                {Mod[Apply[Plus, #], 4] &, {}, 1}, {{1}, 0}, 50]]]
```



This runs 2D 9-neighbor totalistic code 3702 for 25 steps, giving the results for the last 5 steps.

```
In[13] := Show[GraphicsArray[Map[RasterGraphics,
                CellularAutomaton[{3702, {2, 1}, {1, 1}}, {{{1}}, 0}, 25, -5]]]]
```

▪ **Special-purpose hardware.** The simple structure of cellular automata makes it natural to think of implementing them with special-purpose hardware. And indeed from the 1950s on, a sequence of special-purpose machines have been built to implement 1D, 2D and sometimes 3D cellular automata. Two basic ideas have been used: parallelism and pipelines. Both ideas rely on the local nature of cellular automaton rules.

In the parallel approach, the machine has many separate processors, each dedicated to handling a single cell or a small group of cells. In the pipelined approach, there is just a single processor (or perhaps a few processors) through which the data on different cells is successively piped. The key point, however, is that at every stage it is easy to know what data will be needed, so this data can be prefetched, potentially through a specially built memory system.

In general, the speed increases that can be achieved depend on many details of memory and communications architecture. The increases have tended to become less significant over the years, as the on-chip memories of microprocessors have become larger, and the time necessary to send data from one chip to another has become proportionately more important.

In the future, however, new technologies may change the trade-offs, and indeed cellular automata are obvious candidates for early implementation in both nanotechnology and optical computing. (See also page 841.)

▪ **Audio representation.** A step in the evolution of a cellular automaton can be represented as a sound by treating each cell like a key on a piano, with the key taken to be pressed if the cell is black. This yields a chord such as

```
Play[Evaluate[Apply[Plus, Flatten[Map[Sin[1000 # t] &,
    N[2^(1/12)]^Position[list, 1]]]]], {t, 0, 0.2}]
```

A sequence of such chords can sometimes provide a useful representation of cellular automaton evolution. (See also page 1080.)

▪ **Cellular automaton rules as formulas.** The value *a[t, i]* for a cell on step *t* at position *i* in any of the cellular automata in this chapter can be obtained from the definition

```
a[t_, i_] := f[a[t - 1, i - 1], a[t - 1, i], a[t - 1, i + 1]]
```

Different rules correspond to different choices of the function *f*. For example, rule 90 on page 25 corresponds to

```
f[1, _, 1] = 0; f[0, _, 1] = 1; f[1, _, 0] = 1; f[0, _, 0] = 0
```

One can specify initial conditions for example by

```
a[0, 0] = 1; a[0, _] = 0
```

(the cell on step 0 at position 0 has value 1, but all other cells on that step have value 0). Then just asking for *a[4, 0]* one will immediately get the value after 4 steps of the cell at position 0. (For efficiency, the main definition should in practice be given as

```
a[t_, i_] := a[t, i] = f[a[t - 1, i - 1], a[t - 1, i], a[t - 1, i + 1]]
```

so that all intermediate values which are computed are automatically stored.)

The definition of the function *f* for rule 90 that we gave above is essentially just a look-up table. But it is also possible to define this function in an algebraic way

```
f[p_, q_, r_] := Mod[p + r, 2]
```

Algebraic definitions can also be given for other rules:

- Rule 254 (page 24): *1 - (1 - p) (1 - q) (1 - r)*
- Rule 250 (page 25): *p + r - p r*
- Rule 30 (page 27): *Mod[p + q + r + q r, 2]*
- Rule 110 (page 32): *Mod[(1 + p) q r + q + r, 2]*

In these definitions, we represent the values of cells by the numbers 1 or 0. If values +1 and -1 are used instead, different formulas are obtained; rule 90, for example, corresponds to *p r*. It is also possible to represent values of cells as *True* and *False*. And in this case cellular automaton rules become logic expressions:

- Rule 254: *Or[p, q, r]*
- Rule 250: *Or[p, r]*
- Rule 90: *Xor[p, r]*
- Rule 30: *Xor[p, Or[q, r]]*
- Rule 110: *Xor[Or[p, q], And[p, q, r]]*

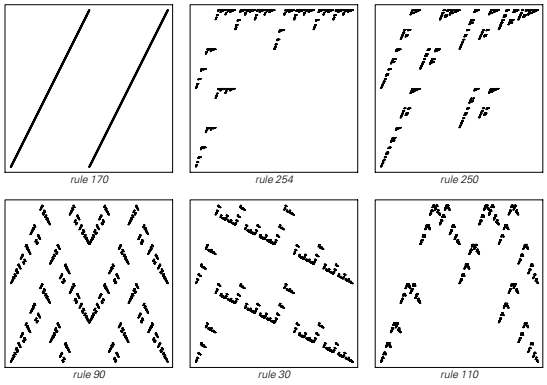
(Note that *Not[p]* corresponds to *1 - p*, *And[p, q]* to *p q*, *Xor[p, q]* to *Mod[p + q, 2]* and *Or[p, q]* to *Mod[p q + p + q, 2]*.)

Given either the algebraic or logical form of a cellular automaton rule, it is possible at least in principle to generate symbolic formulas for the results of cellular automaton evolution. Thus, for example, one can use initial conditions

```
a[0, -1] = p; a[0, 0] = q; a[0, 1] = r; a[0, _] = 0
```

to generate a formula for the value of a cell that holds for any choice of values for the three initial center cells. In practice, however, most such formulas rapidly become very complicated, as discussed on page 618.

▪ **Mathematical interpretation of cellular automata.** In the context of pure mathematics, the state space of a 1D cellular automaton with an infinite number of cells can be viewed as a Cantor set. The cellular automaton rule then corresponds to a continuous mapping of this Cantor set to itself (continuity follows from the locality of the rule). (Compare page 959.)



The pictures above show representations of the mappings corresponding to various rules, obtained by plotting *Sum[a[t + 1, i] 2^-i, {i, -n, n}]* against *Sum[a[t, i] 2^-i, {i, -n, n}]*

for all possible choices of the *a[t, i]*. (Periodic boundary conditions are used, so that the *a[t, i]* can be viewed as corresponding precisely to digits of rational numbers.) Rule 170 is the classic shift map which shifts all cell values one position to the left without changing them. In the pictures below, this map has the form *Mod[2 x, 1]* (compare page 153).

■ **Page 26 · Pascal's triangle and rule 90.** As shown on page 611 the pattern produced by rule 90 is exactly Pascal's triangle of binomial coefficients reduced modulo 2: black cells correspond to odd binomial coefficients.

The number of black cells on row *t* is given by *2^DigitCount[t, 2, 1]*, where *DigitCount[t, 2, 1]* is plotted on page 902. The positions of the black cells are given by (and this establishes the connection with the picture on page 117)

```
Fold[Flatten[{#1 - #2, #1 + #2}] &, 0, 2^DigitPositions[t]]

DigitPositions[n_] :=
  Flatten[Position[Reverse[IntegerDigits[n, 2]], 1]] - 1
```

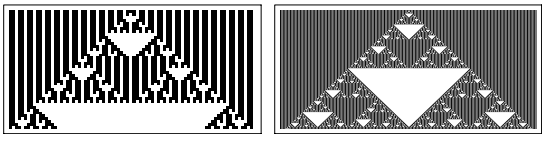
The actual pattern generated by rule 90 corresponds to the coefficients in *PolynomialMod[Expand[(1/x + x)^t], 2]* (see page 1091); the color of a particular cell is thus given by *Mod[Binomial[t, (n + t)/2], 2] /; EvenQ[n + t]*.

Mod[Binomial[t, n], 2] yields a distorted pattern that is the one produced by rule 60 (see page 58). In this pattern, the color of a particular cell can be obtained directly from the digit sequences for *t* and *n* by *1 - Sign[BitAnd[-t, n]]* or (see page 583)

```
With[{d = Ceiling[Log[2, Max[t, n] + 1]]}, If[FreeQ[
    IntegerDigits[t, 2, d] - IntegerDigits[n, 2, d], -1], 1, 0]]
```

■ **Self-similarity.** The pattern generated by rule 90 after a given number of steps has the property that it is identical to what one would get by going twice as many steps, and then keeping only every other row and column. After 2^m steps the triangular region outlined by the pattern contains altogether 4^m cells, but only 3^m of these are black. In the limit of an infinite number of steps one gets a fractal known as a Sierpiński pattern (see page 934), with fractal dimension *Log[2, 3]* ≃ *1.59* (see page 933). Nesting occurs in all cellular automata with additive rules (see page 955).

■ **Another initial condition.** Inserting a single ■ in a background of ■□ blocks in rule 90 yields the pattern below in which both the white and striped regions have fractal dimension 2.



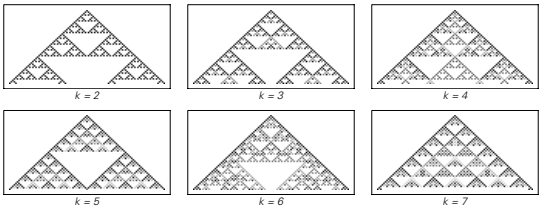
■ **More colors.** The pictures below show generalizations of rule 90 to *k* possible colors using the rule

```
CAStep[k_Integer, a_List] :=
  Mod[RotateLeft[a] + RotateRight[a], k]
```

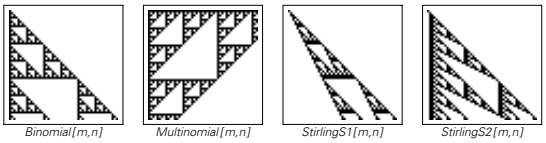
or equivalently *Mod[ListCorrelate[{1, 0, 1}, a, 2], k]*. The number of cells that are not white on row *t* in this case is given by *Apply[Times, 1 + IntegerDigits[t, k]]*. (For non-prime *k*, the patterns are obtained by superimposing the patterns corresponding to the factors of *k*.) A related result is that *IntegerExponent[Binomial[t, n], k]* is given by the number of borrows in the base *k* subtraction of *n* from *t*. *Mod[Binomial[t, n], k]* is given for prime *k* by

```
With[{d = Ceiling[Log[k, Max[t, n] + 1]]},
  Mod[Apply[Times, Apply[Binomial, Transpose[
      {IntegerDigits[t, k, d], IntegerDigits[n, k, d]}], {1}]], k]]
```

The patterns obtained for any *k* are nested. For prime *k* the total number of non-white cells down to step k^m is $(1/2\,k\,(k+1))^m$ and the patterns have fractal dimension *1 + Log[k, (k + 1)/2]* (see page 955). These are examples of additive rules, discussed further on page 952. (See also page 922 for the continuous case.)



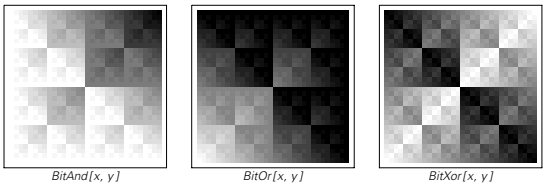
■ **History.** Pascal's triangle probably dates from antiquity; it was known in China in the 1200s, and was discussed in some detail by Blaise Pascal in 1654, particularly in connection with probability theory. The digit-based approach to finding binomial coefficients modulo *k* has been invented independently many times since the mid-1800s, notably by Edouard Lucas in 1877 and James Glaisher in 1899. The fact that the odd binomial coefficients form a nested geometrical pattern had apparently not been widely noticed before I emphasized it in 1982.

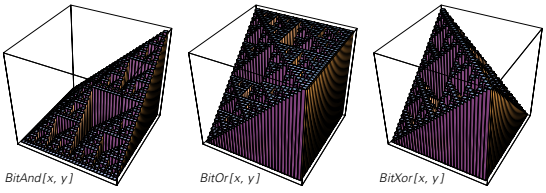


■ **Other integer functions.** The pictures above show patterns produced by reducing several integer functions modulo 2. With *d* arguments *Multinomial* yields a nested pattern in *d* dimensions. Note that *GCD[m, n]* yields a more complicated

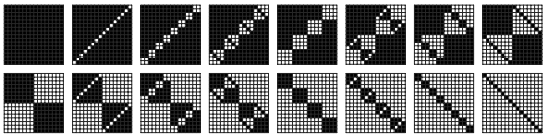
pattern (see page 613), as do *JacobiSymbol[m, 2 n - 1]* (see page 1081) and various combinations of functions (see page 747).

▪ **Bitwise functions.** Bitwise functions typically yield nested patterns. (As discussed above, any cellular automaton rule can be represented as an appropriate combination of bitwise functions.) Note that *BitOr[x, y] + BitAnd[x, y] == x + y* and *BitOr[x, y] - BitAnd[x, y] == BitXor[x, y]*.

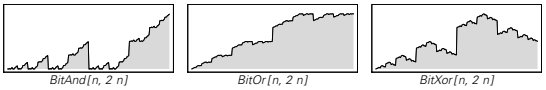




The patterns below show where *BitXor[x, y] == t* for successive *t* and correspond to steps in the "munching squares" program studied on the PDP-1 computer in 1962.

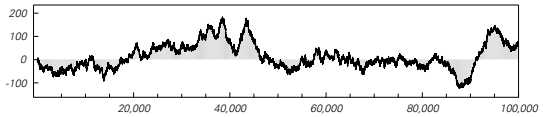


Nesting is also seen in curves obtained by applying bitwise functions to *n* and *2 n* for successive *n*. Note that *2 n* has the same digits as *n*, but shifted one position to the left.



▪ **Page 28 · Tests of randomness.** The statistical tests that I have performed include the eight listed on page 1084.

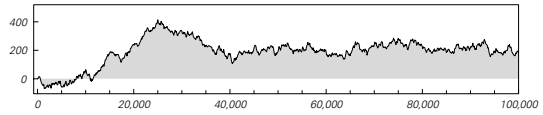
▪ **Page 29 · Rule 30.** The left-hand side of the pattern shown has an obvious repetitive character. In general, if one looks along a diagonal *n* cells in from either edge of the pattern, then the period of repetition can be at most 2^n. On the right-hand edge, the first few periods that are seen are *{1, 2, 2, 4, 8, 8, 16, 32, 32, 64, 64, 64, 64, 128, 256}* and in general the period seems to increase exponentially with depth. On the left-hand edge, the period increases only extremely slowly: period 2 is first achieved at depth 3, period 4 at depth 8, 8 at 29, 16 at 400, 32 at 87,867, 64 at 2,107,985,255 or more, and so on. (Each period doubling turns out to occur exactly when a diagonal in the pattern eventually becomes a white stripe, and the diagonal to its left has an odd number of black cells in each repeating block.) The boundary that separates repetition on the left from randomness on the right moves an average of about 0.252 cells to the left at every step (compare page 949). The picture below shows the fluctuations around this average.

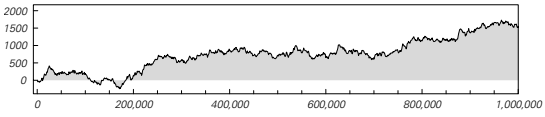


Complete pattern. All possible blocks appear to occur eventually (see page 725). The probability for a block of *n* adjacent white cells (corresponding to a row in a white triangle) seems quite accurately to approach 2^{-n}, with the first length 10 such block occurring at step 67 and the first length 20 one occurring at step 515.

Center column. The pictures below show the excess of black over white cells in the center column. Out of the first 100,000 cells, a total of 50,098 are black, and out of the first million 500,768 are. The longest run of identical colors in the first 100,000 cells consists of 21 black cells, and in the first million elements 22 black cells. The first *n* elements can be found efficiently using

```
Module[{a = 1}, Table[First[IntegerDigits[
    a, a = BitXor[a, BitOr[2 a, 4 a]]; 2, i]], {i, n}]]
```





The sequence does not repeat in at least its first million steps, and I would amazed if it ever repeats, but as of now I know of no rigorous proof of this. (Erica Jen showed in 1986 that no pair of columns can ever repeat, and the arguments on page 1087 suggest that neither can the center column together with occasional neighboring cells.)

▪ **Page 32 · Rule 110.** Many more details of rule 110 are discussed on pages 229 and 675. Localized structures that can occur are shown on page 292. Note that of the 8 cases in

the basic rule for rule 110, only one differs from rule 102—which is a simple additive rule obtained by reflecting rule 60.

The Need for a New Intuition

■ **Reactions of scientists.** Many scientists find the complexity of the pictures in this chapter so surprising that at first they assume it cannot be real. Typically they imagine that while the pictures may look complicated, they would actually seem simple if only they were subjected to the appropriate kind of analysis. In Chapter 10 I will give extensive evidence that this is not the case. But suffice it to say here that when it comes to finding regularities even the most advanced methods from mathematics and statistics tend to be no more powerful than our eyes. And whatever formal definition one may use for complexity (see page 557), the fact that our eyes perceive it in the systems discussed in this chapter is already very significant.

■ **Intuition from practical computing.** Everyday experience with computers and programming leads to observations like the following:

- General-purpose computers and general-purpose programming languages can be built.
- Different programs for doing all sorts of different things can be set up.
- Any given program can be implemented in many ways.
- Programs can behave in complicated and seemingly random ways—particularly when they are not working properly.
- Debugging a program can be difficult.
- It is often difficult to foresee what a program can do by reading its code.
- The lower the level of representation of the code for a program the more difficult it tends to be to understand.
- Some computational problems are easy to state but hard to solve.
- Programs that simulate natural systems are among the most computationally expensive.
- It is possible for people to create large programs—at least in pieces.
- It is almost always possible to optimize a program more, but the optimized version may be more difficult to understand.
- Shorter programs are sometimes more efficient, but optimizations often require many cases to be treated separately, making programs longer.
- If programs are patched too much, they typically stop working at all.

■ **Applications to design.** Many of the pictures in this book look strikingly similar to artistic designs of various styles. Probably this reflects not so much a similarity in underlying rules, but rather similarity in features that are most noticeable to the human visual system. Note that square grids of colored cells as in the cellular automata in this chapter can be used quite directly as weaving patterns. (See also page 929.)

Why These Discoveries Were Not Made Before

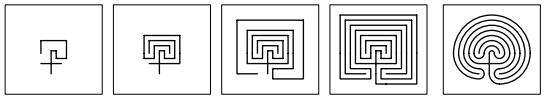
■ **Page 43 · Ornamental art.** Almost all major cultural periods are associated with certain characteristic forms of ornament. Often the forms of ornament used on particular kinds of objects probably arose as idealized imitations of earlier or more natural forms for such objects—so that, for example, imitations of weaving, bricks and various plant forms are common. Large-scale purely abstract patterns were also central to art in such cultural traditions as Islam where natural forms were considered works of God that must not be shown directly. Once established, styles of ornament tend to be repeated extensively as a way of providing certain comfort and familiarity—especially in architecture. The vast majority of elaborate ornament seems to have been created by artisans with little or no formal theoretical discussion, although particularly since the 1800s there have been various attempts to find systematic ways to catalog forms of ornament, sometimes based on analogies with grammar. (Issues of proportion have however long been the subject of considerable theoretical discussion.) It is notable that whereas repetitive patterns have been used extensively in ornament, even nesting is rather rare. And even though for example elaborate symmetry rules have been devised, nothing like cellular automaton rules appear to have ever arisen. The results in this book now show that such rules can capture the essence of many complex processes that occur in nature—so that even though they lack historical context such rules can potentially provide a basis for forms of ornament that are familiar as idealizations of nature. (Compare page 929.)

The pictures in the main text show a sequence of early examples of various characteristic forms of ornament.

22,000 BC (Paleolithic). Mammoth ivory bracelet from Mezin, Ukraine. Similar zig-zag designs are seen in other objects from the same period. In the example shown, it is notable that the angle of the zig-zags is comparable to the angle of the Schreger lines that occur naturally in mammoth dentin.

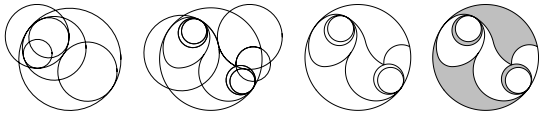
3000 BC (Sumerian). Columns with three colors of clay pegs set in mud from a wall of the Eanna temple in Uruk, Mesopotamia (Warka, Iraq)—perhaps mentioned in the Epic of Gilgamesh. (Now in the Staatliche Museum, Berlin.) This is the earliest known explicit example of mosaic.

1200 BC (Greek). The back of a clay accounting tablet from Pylos, Greece. The pattern was presumably made by the procedure shown below. Legend has it that it was the plan for the labyrinth housing the minotaur in the palace at Knossos, Crete, and that it was designed by Daedalus. It is also said that it was a logo for the city of Troy—or perhaps the plan of some of its walls. The pattern—in either its square or rounded form—has appeared with remarkably little variation in a huge variety of places all over the world—from Cretan coins, to graffiti at Pompeii, to the floor of the cathedral at Chartres, to carvings in Peru, to logos for aboriginal tribes. For probably three thousand years, it has been the single most common design used for mazes.

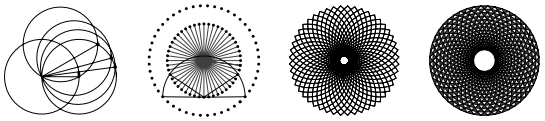


900 BC (Phoenician). Ivory carving presumably from the Mediterranean area. (Now in the British Museum.) This was a common decorative pattern, formed by drawing circles centered at holes arranged in a triangular array. It is also found in Egyptian and other art. Such patterns were discussed by Euclid and later Leonardo da Vinci in connection with the theory of lunes.

1st century BC (Celtic). The back of the so-called Desborough Mirror—a bronze mirror from Desborough, England made in the Iron Age sometime between 50 BC and 50 AD. (Now in the British Museum.) The engraved pattern is made of parts of circles that just touch each other, as in the picture below.



2nd century AD (Roman). A mosaic from a complex in Rome, Italy. (Now in the National Museum, Rome.) The geometrical pattern was presumably made by first constructing 48 regularly spaced spokes by repeated angle bisection, as in the first picture below, then drawing semicircles centered at the end of each spoke, and finally adding concentric circles through the intersection points. Similar rosette patterns may have been used in Greece around 350 BC; they became popular in churches in the 1500s.



8th century (Islamic). A detail on the outside wall of the Great Mosque of Córdoba, Spain, built around 785 AD.

8th century (Celtic). An area less than 2 inches square from inside the letter ρ on the extremely elaborate chi-rho page of the Book of Kells, an illuminated gospel manuscript created over a period of years at various monasteries, probably starting around 800 AD at the Irish monastery on the island of Iona, Scotland. Even on this one page there are perhaps a dozen other very similar nested structures.

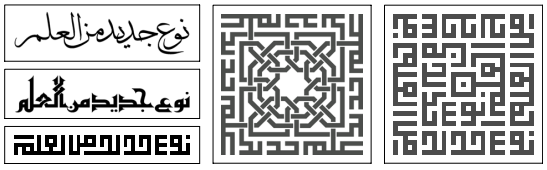
12th century (Italian). A window in the Palatine Chapel in Palermo, Sicily, presumably built around 1140 AD. The chapel is characteristic of so-called Arab-Norman style.

13th century (English). The Dean's Eye rose window of the Lincoln cathedral in England, built around 1225 AD. Similar tree-like patterns are seen in many Gothic windows from the same general period.

13th century (Italian) (4 pictures). Marble mosaics on the floor of the cathedral at Anagni, Italy, made around 1226 AD by Cosmas of the Cosmati group. (The fourth picture is a close-up of the third.) The third picture—particularly the part magnified in the fourth picture—shows an approximate nested structure, presumably created as in the pictures below. The triangles are all equilateral, with the result that at a given step several different sizes of triangles occur—though the basic structure of the pattern is still the same as from the rule 90 cellular automaton. (Compare the Apollonian packing of page 986.) The Cosmati group—mostly four generations of one family—made elaborate geometrical and other mosaics with a mixture of Byzantine, Islamic and other influences from about 1190 to 1300, mostly in and around Rome, but also for example in Westminster Abbey in England. Triangular shapes with one level of nesting are quite common in their work; three levels of nesting as shown here

are rare. It is notable that in later imitations of Cosmati mosaics, these kinds of patterns were almost never used.

14th century (Islamic). Wall decoration in the Pir-i-Bakran mausoleum in Linjan, Iran, built around 1299–1312. The pattern is square Kufi calligraphy for a widely quoted verse of the Koran. Starting from traditional Naskhi Arabic script, as in the picture below, the Kufi style began to develop around 900 AD, with square Kufi being used in architectural ornamental by about 1100 AD.



14th century (Islamic). Tiled wall in the Alcázar of Seville, Spain, built in 1364. (The same pattern was used at about the same time in the Alhambra in Granada, Spain.) The pattern can be made by starting with a grid of triangles, then consistently pushing in or out the sides of each one. (Notable uses of such patterns were made by Maurits Escher starting in the 1930s.)

Other cases. The cases that are known inevitably tend to be ones created out of stone or ceramic materials that survive; no doubt there were others created for example with wood or textiles. One case with wood is Chinese lattice. What has survived mostly shows repetitive patterns, but the ice-ray style, probably going back to 100 AD, has approximate nesting, though with many random elements. The patterns shown are all basically two-dimensional. An example of 1D ornamental patterns are molding profiles. Ever since antiquity these have often been quite elaborate, and it is conceivable that they can sometimes be interpreted as showing nesting.

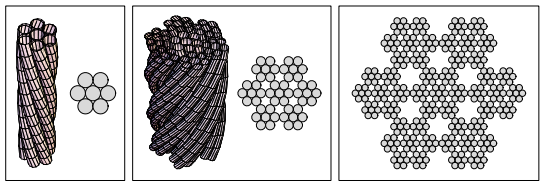
▪ **Recognition of art.** One bizarre possibility is that forms like those from rule 30 could have been created as art long ago but not be recognized now. For while it is easy to tell that a cave painting of an animal is a piece of purposeful art, dots carved into a rock in an approximate rule 30 pattern might not even be noticed as something of human origin. But although there are many seemingly random painted patterns in caves from perhaps 30,000 BC, I would be amazed if any of them were actually produced by definite simple rules. (See page 839.)

▪ **The concept of rules.** Processes based on rules occur in a great many areas of human endeavor. Sometimes the rules serve mainly as a constraint. But it is not uncommon for them to be used—like in a cellular automaton—as a way of specifying how structures should be built up. Almost without exception, however, the rules have in the past been chosen to yield only rather specific and simple results. Beyond ornamental art, examples with long histories include:

Architecture. Structures such as ziggurats and pyramids were presumably constructed by assembling collections of stones according to simple rules. The Great Pyramid in Egypt was built around 2500 BC and contains about two million large stones. (By comparison, the pictures of rule 30 on pages 29 and 30 contain a total of about a million cells.) Starting perhaps as long ago as 1000 BC Hindu temples were constructed with similar elements on different scales, yielding a form of approximate nesting. In Roman and later architecture, rooms in buildings have quite often been arranged in roughly nested patterns (an extreme example being the Castel del Monte from the 1200s). From the Middle Ages many Persian gardens (such as those of the Taj Mahal from around 1650) have had fairly regular nested structures obtained by a few successive fourfold subdivisions. And starting in the early 1200s, Gothic windows were often constructed with levels of roughly tree-like nested forms (see above). Nesting does not appear to have been used in physical city plans (except to a small extent in Vauban star fortifications), though it is common in organizational structures. (As indicated above, architectural ornament has also often in effect been constructed using definite rules.)

Textile making. Since early in human history there appear to have been definite rules used for weaving. But insofar as the purpose is to produce fabric the basic arrangement of threads is normally always repetitive.

Rope. Since at least 3000 BC rope has been made by twisting together strands themselves made by twisting, yielding cross-sections with some nesting, as in the second picture below. (Since the development of wire rope in the 1870s precise designs have been used, including at least recently the 7×7×7 one shown last below.)



Knots and string figures. For many thousands of years definite rules have been used for tying knots and presumably also for making string figures. But when the rules have more than a few steps they tend to be repetitive.

Paperfolding. Although paperfolding has presumably been practiced for at least 2000 years, even the nested form on page 892 seems to have been noticed only very recently.

Mathematics. Ever since Babylonian times arithmetic has been done by repeatedly applying simple rules to digits in numbers. And ever since ancient Greek times iterative methods have been used to construct geometrical figures. In the late 1600s the idea also emerged that mathematical proofs could be thought of as consisting of repeated applications of definite rules. But the idea of studying possible simple rules independent of their purpose in generating results seems never to have arisen. And as mathematics began to focus on continuous systems the notion of enumerating possible rules became progressively more difficult.

Logic. Rules of logic have been used since around 400 BC. But beyond forms like syllogisms little seems to have been studied in the way of generating identifiable patterns from them. (See page 1099.)

Grammar. The idea that human language is constructed from words according to definite grammatical rules has existed since at least around perhaps 500 BC when Panini gave a grammar for Sanskrit. (Less formal versions of the idea were also common in ancient Greek times.) But for the most part it was not until about the 1950s that rules of grammar began to be viewed as specifications for generating structures, rather than just constraints. (See page 1103.)

Poetry. Definite rules for rhythm in poetry were already well developed in antiquity—and by perhaps 200 BC Indian work on enumerating their possible forms appears to have led to both Pascal's triangle and Fibonacci numbers. Patterns of rhyme involving iterated length-6 permutations (sestina) and interleaved repetitive sequences (terza rima) were in use by the 1300s, notably by Dante.

Music. Simple progressions and various forms of repetition have presumably been used in music since at least the time of Pythagoras. Beginning in the 1200s more complex forms of interleaving such as those of canons have occasionally been used. And in the past century a few composers have implicitly or explicitly used structures based on simple Fibonacci and other substitution systems. Note that rules such as those of counterpoint are used mainly as constraints, not as ways of generating structure.

Military drill. The notion of using definite rules to organize and maneuver formations of soldiers appears to have existed in Babylonian and Assyrian times, and to be well codified by Roman times. Fairly elaborate cases were described for example by Niccolò Machiavelli in 1521, but all were set up to yield only rather simple behavior, such as a column of soldiers being rearranged into lines. (See the firing squad problem on page 1035.)

Games. Games are normally based on definite rules, but are set up so that at each step they involve choosing one of many possibilities, either by skill or randomness. The game of Go, which originated before 500 BC and perhaps as early as 2300 BC, is a case where particularly simple rules manage to allow remarkably complex patterns of play to occur. (Go involves putting black and white stones on a grid, making it visually similar to a cellular automaton.)

Puzzles. Geometric and arithmetic puzzles surprisingly close to those common today seem to have existed since as long ago as 2000 BC. Usually they are based on constraints, and occasionally they can be thought of as providing evidence that simple constraints can have complicated solutions.

Cryptography. Rules for encrypting messages have been used since perhaps 2000 BC, with non-trivial repetitive schemes becoming common in the 1500s, but more complex schemes not appearing until well into the 1900s. (See page 1085.)

Maze designs. From antiquity until about the 1500s the majority of mazes followed a small number of designs—most often based directly on the one shown on page 873, or with subunits like it. (It is now known that there are many other designs that are also possible.)

Rule-based pictures. It is rather common for geometric doodles to be based on definite rules, but it is rare for the rules to be carried far, or for the doodles to be preserved. Some of Leonardo da Vinci's planned book on "Geometrical Play" from the early 1500s has, however, survived, and shows elaborate patterns satisfying particular constraints. Various attempts to enumerate all possible patterns of particular simple kinds have been made—a notable example being Sébastien Truchet in 1704 drawing 2D patterns formed by combining ◹, ◪, ◥, ◪ in various possible ways.

▪ **Page 44 · Understanding nature.** In Greek times it was noted that simple geometrical rules could explain many features of astronomy—the most obvious being the apparent revolution of the stars and the circular shapes of the Sun and Moon. But it was noted that with few exceptions—like beehives—natural objects that occur terrestrially did not appear to follow any simple geometrical rules. (The most complicated curves in Greek geometry were things like cissoids and conchoids.) So from this it was concluded that only certain supposedly perfect objects like the heavenly bodies could be

expected to be fully amenable to human understanding. What rules for natural objects might in effect have been tried in the Judeo-Christian tradition is less clear—though for example the Book of Job does comment on the difficulty of "numbering the clouds by wisdom". And with the notable exception of the alchemists it continued to be believed throughout the Middle Ages that the wonders of nature were beyond human understanding.

▪ **Atomism.** The idea that everything might be made up from large numbers of discrete elements was discussed around perhaps 450 BC by Leucippus and Democritus. Sometime later the Epicureans then suggested that a few types of elements might suffice, and an analogy was made (notably by Lucretius around 100 AD) to the fact that different configurations of letters can make up all the words in a language. But only some schools of Greek philosophy ever supported atomism, and it soon fell out of favor. It was revived in the late 1600s, when corpuscular theories of both light and matter began to be widely discussed. In the early 1800s arguments based on atoms led to success in chemistry, and in the late 1800s statistical mechanics of large assemblies of atoms were used to explain properties of matter (see page 1019). With the rise of quantum theory in the early 1900s it became firmly established that physical systems contain discrete particles. But it was normally assumed that one should think only about explicit particles with realistic mechanical properties—so that abstract idealizations like cellular automata did not arise. (See also pages 1027 and 1043.)

▪ **History of cellular automata.** Despite their very simple construction, nothing like general cellular automata appear to have been considered before about the 1950s. Yet in the 1950s—inspired in various ways by the advent of electronic computers—several different kinds of systems equivalent to cellular automata were independently introduced. A variety of precursors can be identified. Operations on sequences of digits had been used since antiquity in doing arithmetic. Finite difference approximations to differential equations began to emerge in the early 1900s and were fairly well known by the 1930s. And Turing machines invented in 1936 were based on thinking about arbitrary operations on sequences of discrete elements. (Notions in physics like the Ising model do not appear to have had a direct influence.)

The best-known way in which cellular automata were introduced (and which eventually led to their name) was through work by John von Neumann in trying to develop an abstract model of self-reproduction in biology—a topic which had emerged from investigations in cybernetics. Around 1947—perhaps based on chemical engineering—von Neumann began by thinking about models based on 3D factories described by partial differential equations. Soon he changed to thinking about robotics and imagined perhaps implementing an example using a toy construction set. By analogy to electronic circuit layouts he realized however that 2D should be enough. And following a 1951 suggestion from Stanislaw Ulam (who may have already independently considered the problem) he simplified his model and ended up with a 2D cellular automaton (he apparently hoped later to convert the results back to differential equations). The particular cellular automaton he constructed in 1952–3 had 29 possible colors for each cell, and complicated rules specifically set up to emulate the operations of components of an electronic computer and various mechanical devices. To give a mathematical proof of the possibility of self-reproduction, von Neumann then outlined the construction of a 200,000 cell configuration which would reproduce itself (details were filled in by Arthur Burks in the early 1960s). Von Neumann appears to have believed—presumably in part from seeing the complexity of actual biological organisms and electronic computers—that something like this level of complexity would inevitably be necessary for a system to exhibit sophisticated capabilities such as self-reproduction. In this book I show that this is absolutely not the case, but with the intuition he had from existing mathematics and engineering von Neumann presumably never imagined this.

Two immediate threads emerged from von Neumann's work. The first, mostly in the 1960s, was increasingly whimsical discussion of building actual self-reproducing automata—often in the form of spacecraft. The second was an attempt to capture more of the essence of self-reproduction by mathematical studies of detailed properties of cellular automata. Over the course of the 1960s constructions were found for progressively simpler cellular automata capable of self-reproduction (see page 1179) and universal computation (see page 1115). Starting in the early 1960s a few rather simple general features of cellular automata thought to be relevant to self-reproduction were noticed—and were studied with increasingly elaborate technical formalism. (An example was the so-called Garden of Eden result that there can be configurations in cellular automata that arise only as initial conditions; see page 961.) There were also various explicit constructions done of cellular automata whose behavior showed particular simple features perhaps relevant to self-reproduction (such as so-called firing squad synchronization, as on page 1035).

By the end of the 1950s it had been noted that cellular automata could be viewed as parallel computers, and particularly in the 1960s a sequence of increasingly detailed and technical theorems—often analogous to ones about

Turing machines—were proved about their formal computational capabilities. At the end of the 1960s there then began to be attempts to connect cellular automata to mathematical discussions of dynamical systems—although as discussed below this had in fact already been done a decade earlier, with different terminology. And by the mid-1970s work on cellular automata had mostly become quite esoteric, and interest in it largely waned. (Some work nevertheless continued, particularly in Russia and Japan.) Note that even in computer science various names for cellular automata were used, including tessellation automata, cellular spaces, iterative automata, homogeneous structures and universal spaces.

As mentioned in the main text, there were by the late 1950s already all sorts of general-purpose computers on which simulations of cellular automata would have been easy to perform. But for the most part these computers were used to study traditional much more complicated systems such as partial differential equations. Around 1960, however, there were a couple of simulations related to 2D cellular automata done. Stanislaw Ulam and others used computers at Los Alamos to produce a handful of examples of what they called recursively defined geometrical objects—essentially the results of evolving generalized 2D cellular automata from single black cells (see page 928). Especially after obtaining larger pictures in 1967, Ulam noted that in at least one case fairly simple growth rules generated a complicated pattern, and mentioned that this might be relevant to biology. But perhaps because almost no progress was made on this with traditional mathematical methods, the result was not widely known, and was never pursued. (Ulam tried to construct a 1D analog, but ended up not with a cellular automaton, but instead with the sequences based on numbers discussed on page 908.) Around 1961 Edward Fredkin simulated the 2D analog of rule 90 on a PDP-1 computer, and noted its self-reproduction properties (see page 1179), but was generally more interested in finding simple physics-like features.

Despite the lack of investigation in science, one example of a cellular automaton did enter recreational computing in a major way in the early 1970s. Apparently motivated in part by questions in mathematical logic, and in part by work on "simulation games" by Ulam and others, John Conway in 1968 began doing experiments (mostly by hand, but later on a PDP-7 computer) with a variety of different 2D cellular automaton rules, and by 1970 had come up with a simple set of rules he called "The Game of Life", that exhibit a range of complex behavior (see page 249). Largely through popularization in *Scientific American* by Martin Gardner, Life became widely known. An immense amount of effort was spent finding special initial conditions that give particular forms of repetitive or other behavior, but virtually no systematic scientific work was done (perhaps in part because even Conway treated the system largely as a recreation), and almost without exception only the very specific rules of Life were ever investigated. (In 1978 as a possible 1D analog of Life easier to implement on early personal computers Jonathan Millen did however briefly consider what turns out to be the code 20 $k = 2$, $r = 2$ totalistic rule from page 283.)

Quite disconnected from all this, even in the 1950s, specific types of 2D and 1D cellular automata were already being used in various electronic devices and special-purpose computers. In fact, when digital image processing began to be done in the mid-1950s (for such applications as optical character recognition and microscopic particle counting) 2D cellular automaton rules were usually what was used to remove noise. And for several decades starting in 1960 a long line of so-called cellular logic systems were built to implement 2D cellular automata, mainly for image processing. Most of the rules used were specifically set up to have simple behavior, but occasionally it was noted as a largely recreational matter that for example patterns of alternating stripes ("custering") could be generated.

In the late 1950s and early 1960s schemes for electronic miniaturization and early integrated circuits were often based on having identical logical elements laid out on lines or grids to form so-called cellular arrays. In the early 1960s there was for a time interest in iterative arrays in which data would be run repeatedly through such systems. But few design principles emerged, and the technology for making chips with more elaborate and less uniform circuits developed rapidly. Ever since the 1960s the idea of making array or parallel computers has nevertheless resurfaced repeatedly, notably in systems like the ILLIAC IV from the 1960s and 1970s, and systolic arrays and various massively parallel computers from the 1980s. Typically the rules imagined for each element of such systems are however immensely more complicated than for any of the simple cellular automata I consider.

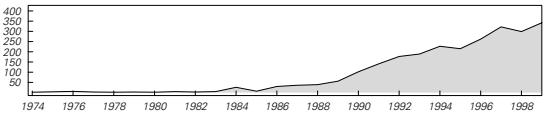
From at least the early 1940s, electronic or other digital delay lines or shift registers were a common way to store data such as digits of numbers, and by the late 1940s it had been noted that so-called linear feedback shift registers (see page 974) could generate complicated output sequences. These systems turn out to be essentially 1D additive cellular automata (like rule 90) with a limited number of cells (compare page 259). Extensive algebraic analysis of their behavior was done starting in the mid-1950s, but most of it concentrated on issues like repetition periods, and did not even explicitly

uncover nested patterns. (Related analysis of linear recurrences over finite fields had been done in a few cases in the 1800s, and in some detail in the 1930s.) General 1D cellular automata are related to nonlinear feedback shift registers, and some explorations of these—including ones surprisingly close to rule 30 (see page 1088)—were made using special-purpose hardware by Solomon Golomb in 1956–9 for applications in jamming-resistant radio control—though again concentrating on issues like repetition periods. Linear feedback shift registers quickly became widely used in communications applications. Nonlinear feedback shift registers seem to have been used extensively for military cryptography, but despite persistent rumors the details of what was done continue to be secret.

In pure mathematics, infinite sequences of 0's and 1's have been considered in various forms since at least the late 1800s. Starting in the 1930s the development of symbolic dynamics (see page 960) led to the investigation of mappings of such sequences to themselves. And by the mid-1950s studies were being made (notably by Gustav Hedlund) of so-called shift-commuting block maps—which turn out to be exactly 1D cellular automata (see page 961). In the 1950s and early 1960s there was work in this area (at least in the U.S.) by a number of distinguished pure mathematicians, but since it was in large part for application to cryptography, much of it was kept secret. And what was published was mostly abstract theorems about features too global to reveal any of the kind of complexity I discuss.

Specific types of cellular automata have also arisen—usually under different names—in a vast range of situations. In the late 1950s and early 1960s what were essentially 1D cellular automata were studied as a way to optimize circuits for arithmetic and other operations. From the 1960s onward simulations of idealized neural networks sometimes had neurons connected to neighbors on a grid, yielding a 2D cellular automaton. Similarly, various models of active media—particularly heart and other muscles—and reaction-diffusion processes used a discrete grid and discrete excitation states, corresponding to a 2D cellular automaton. (In physics, discrete idealizations of statistical mechanics and dynamic versions of systems like the Ising model were sometimes close to cellular automata, except for the crucial difference of having randomness built into their underlying rules.) Additive cellular automata such as rule 90 had implicitly arisen in studies of binomial coefficient modulo primes in the 1800s (see page 870), but also appeared in various settings such as the "forests of stunted trees" studied around 1970.

Yet by the late 1970s, despite all these different directions, research on systems equivalent to cellular automata had largely petered out. That this should have happened just around the time when computers were first becoming widely available for exploratory work is ironic. But in a sense it was fortunate, because it allowed me when I started working on cellular automata in 1981 to define the field in a new way (though somewhat to my later regret I chose—in an attempt to recognize history—to use the name "cellular automata" for the systems I was studying). The publication of my first paper on cellular automata in 1983 (see page 881) led to a rapid increase of interest in the field, and over the years since then a steadily increasing number of papers (as indicated by the number of source documents in the Science Citation Index shown below) have been published on cellular automata—almost all following the directions I defined.



▪ **Close approaches.** The basic phenomena in this chapter have come at least somewhat close to being discovered many times in the past. The historical progression of primary examples of this seem to be as follows:

- 500s–200s BC: Simply-stated problems such as finding primes or perfect numbers are presumably seen to have complicated solutions, but no general significance is attached to this (see pages 132 and 910).
- 1200s: Fibonacci sequences, Pascal's triangle and other rule-based numerical constructions are studied, but are found to show only simple behavior.
- 1500s: Leonardo da Vinci experiments with rules corresponding to simple geometrical constraints (see page 875), but finds only simple forms satisfying these constraints.
- 1700s: Leonhard Euler and others compute continued fraction representations for numbers with simple formulas (see pages 143 and 915), noting regularity in some cases, but making no comment in other cases.
- 1700s and 1800s: The digits of π and other transcendental numbers are seen to exhibit apparent randomness (see page 136), but the idea of thinking about this randomness as coming from the process of calculation does not arise.
- 1800s: The distribution of primes is studied extensively—but mostly its regularities, rather than its irregularities, are considered. (See page 132.)

- 1800s: Complicated behavior is found in the three-body problem, but it is assumed that with better mathematical techniques it will eventually be resolved. (See page 972.)
- 1880s: John Venn and others note the apparent randomness of the digits of π, but somehow take it for granted.
- 1906: Axel Thue studies simple substitution systems (see page 893) and finds behavior that seems complicated—though it turns out to be nested.
- 1910s: Gaston Julia and others study iterated maps, but concentrate on properties amenable to simple description.
- 1920: Moses Schönfinkel introduces combinators (see page 1121) but considers mostly cases specifically constructed to correspond to ordinary logical functions.
- 1921: Emil Post looks at a simple tag system (see page 894) whose behavior is difficult to predict, but failing to prove anything about it, goes on to other problems.
- 1920: The Ising model is introduced, but only statistics of configurations, and not any dynamics, are studied.
- 1931: Kurt Gödel establishes Gödel's Theorem (see page 782), but the constructions he uses are so complicated that he and others assume that simple systems can never exhibit similar phenomena.
- Mid-1930s: Alan Turing, Alonzo Church, Emil Post, etc. introduce various models of computation, but use them in constructing proofs, and do not investigate the actual behavior of simple examples.
- 1930s: The $3n+1$ problem (see page 904) is posed, and unpredictable behavior is found, but the main focus is on proving a simple result about it.
- Late 1940s and 1950s: Pseudorandom number generators are developed (see page 974), but are viewed as tricks whose behavior has no particular scientific significance.
- Late 1940s and early 1950s: Complex behavior is occasionally observed in fairly simple electronic devices built to illustrate ideas of cybernetics, but is usually viewed as something to avoid.
- 1952: Alan Turing applies computers to studying biological systems, but uses traditional mathematical models rather than, say, Turing machines.
- 1952–1953: John von Neumann makes theoretical studies of complicated cellular automata, but does not try looking at simpler cases, or simulating the systems on a computer.
- Mid-1950s: Enrico Fermi and collaborators simulate simple systems of nonlinear springs on a computer, but do not notice that simple initial conditions can lead to complicated behavior.
- Mid-1950s to mid-1960s: Specific 2D cellular automata are used for image processing; a few rules showing slightly complex behavior are noticed, but are considered of purely recreational interest.
- Late 1950s: Computer simulations of iterated maps are done, but concentrate mostly on repetitive behavior. (See page 918.)
- Late 1950s: Ideas from dynamical systems theory begin to be applied to systems equivalent to 1D cellular automata, but details of specific behavior are not studied except in trivial cases.
- Late 1950s: Idealized neural networks are simulated on digital computers, but the somewhat complicated behavior seen is considered mainly a distraction from the phenomena of interest, and is not investigated. (See page 1099.)
- Late 1950s: Berni Alder and Thomas Wainwright do computer simulations of dynamics of hard sphere idealized molecules, but concentrate on large-scale features that do not show complexity. (See page 999.)
- 1956–1959: Solomon Golomb simulates nonlinear feedback shift registers—some with rules close to rule 30—but studies mainly their repetition periods not their detailed complex behavior. (See page 1088.)
- 1960, 1967: Stanislaw Ulam and collaborators simulate systems close to 2D cellular automata, and note the appearance of complicated patterns (see above).
- 1961: Edward Fredkin simulates the 2D analog of rule 90 and notes features that amount to nesting (see above).
- Early 1960s: Students at MIT try running many small computer programs, and in some cases visualizing their output. They discover various examples (such as "munching foos") that produce nested behavior (see page 871), but do not go further.
- 1962: Marvin Minsky and others study many simple Turing machines, but do not go far enough to discover the complex behavior shown on page 81.
- 1963: Edward Lorenz simulates a differential equation that shows complex behavior (see page 971), but concentrates on its lack of periodicity and sensitive dependence on initial conditions.
- Mid-1960s: Simulations of random Boolean networks are done (see page 936), but concentrate on simple average properties.

- 1970: John Conway introduces the Game of Life 2D cellular automaton (see above).
- 1971: Michael Paterson considers a class of simple 2D Turing machines that he calls worms and that exhibit complicated behavior (see page 930).
- 1973: I look at some 2D cellular automata, but force the rules to have properties that prevent complex behavior (see page 864).
- Mid-1970s: Benoit Mandelbrot develops the idea of fractals (see page 934), and emphasizes the importance of computer graphics in studying complex forms.
- Mid-1970s: Tommaso Toffoli simulates all 4096 2D cellular automata of the simplest type, but studies mainly just their stabilization from random initial conditions.
- Late 1970s: Douglas Hofstadter studies a recursive sequence with complicated behavior (see page 907), but does not take it far enough to conclude much.
- 1979: Benoit Mandelbrot discovers the Mandelbrot set (see page 934) but concentrates on its nested structure, not its overall complexity.
- 1981: I begin to study 1D cellular automata, and generate a small picture analogous to the one of rule 30 on page 27, but fail to study it.
- 1984: I make a detailed study of rule 30, and begin to understand the significance of it and systems like it.

▪ **The importance of explicitness.** Looking through this book, one striking difference with most previous scientific accounts is the presence of so many explicit pictures that show how every element in a system behaves. In the past, people have tended to consider it more scientific to give only numerical summaries of such data. But most of the phenomena I discuss in this book could not have been found without such explicit pictures. (See also page 108.)

▪ **My work on cellular automata.** I began serious work on cellular automata in the middle of 1981. I had been thinking for some time about how complicated patterns could arise in natural systems—in apparent violation of the Second Law of Thermodynamics. I had been particularly interested in self-gravitating gases where the basic physics seemed clear, but where complex phenomena like galaxy formation seemed to occur. I had also been interested in neural networks, where there had been fairly simple models developed by Warren McCulloch and Walter Pitts in the 1940s. I came up with cellular automata as an attempt to capture the essential features of a range of systems, from self-gravitating gases to neural networks. I wanted to find models that had a simple structure like the Ising model in statistical mechanics (studied since the 1920s), but which had definite rules for time evolution and could easily be simulated on a computer. Ironically enough, while cellular automata are good for many things, they turn out to be rather unsuitable for modelling either self-gravitating gases or neural networks. (See page 1021). But by the time I realized this, it was clear that cellular automata were of great interest for many other purposes.

I did my first major computer experiments on cellular automata late in 1981 (see page 19). Two features initially struck me most. First, that starting from random initial conditions, cellular automata could organize themselves to produce complex patterns. And second, that in cases like rule 90 simple initial conditions led to nested or fractal patterns. During the first half of 1982, I worked hard to analyze the behavior of cellular automata using ideas from statistical mechanics, dynamical systems theory and discrete mathematics. And in June 1982, I finished my first paper on cellular automata, entitled "Statistical Mechanics of Cellular Automata". Published in the journal *Reviews of Modern Physics* in July 1983, this paper already presents in raw form many of the key ideas that led to the development of the science described in this book. It discusses the fact that by not using traditional mathematical equations, simple models can potentially be made to reproduce complex phenomena, and it mentions some of the consequences of viewing models like cellular automata as computational systems. The paper also contained a small picture of rule 30 started from a single black cell. But at the time, I did not study this picture in detail, and I tacitly assumed that whenever I saw randomness it must come from the random initial conditions that I used. (See page 112.)

It was some time in the fall of 1981 that I first found out (at a dinner with some then-young MIT computer scientists) that a version of the systems I had invented had been studied before under the name of "cellular automata". (I had been aware of the Game of Life, but its recreational emphasis had put me off studying it.) Knowing the name cellular automata, I was able to track down quite a number of relevant papers from the 1950s and 1960s. But I found that active research on what had been called cellular automata had more or less petered out (with the slight exception of a group at MIT at that time mainly concerned with building special-purpose hardware for 2D cellular automata). By late 1982 preprints of my paper on cellular automata had created quite a stir, and I got involved in organizing a conference held in March 1983 at Los Alamos to bring together many people newly interested in cellular automata with earlier workers in the field.

As part of preparing for that conference, I decided to use the graphics capabilities of the new workstation computer I had just obtained (a very early unit from Sun Microsystems) to investigate in a systematic way the behavior of a large collection of different cellular automata. And after spending several weeks looking at screen after screen of patterns—and trying to analyze their properties—I came to the conclusion that one could identify in the behavior of cellular automata with random initial conditions just four basic classes, each with its own characteristic features (see page 231).

In 1982 and early 1983, my efforts to analyze cellular automata were mainly based on ideas from discrete mathematics and dynamical systems theory. In the course of 1983, I also began to make serious use of formal language theory and the theory of computation. But for the most part I concentrated on characterizing behavior obtained from all possible initial conditions. And in fact I still vaguely assumed that if simple initial conditions were used, only fairly simple behavior would be obtained. Several of my papers had actually shown quite detailed pictures where this was not the case. I had noticed them, but they had never been among the examples I had studied in depth, partly for the superficial reason that the rules they involved were not symmetrical, or inevitably led to patterns that were otherwise not convenient for display. I do not know exactly what made me start looking more carefully at simple initial conditions, though I believe that I first systematically generated high-resolution pictures of all the $k = 2$, $r = 1$ cellular automata as an exercise for an early laserprinter—probably at the beginning of 1984. And I do know that for example on June 1, 1984 I printed out pictures of rule 30, rule 110 and $k = 2$, $r = 2$ totalistic code 10 (see note below), took them with me on a flight from New York to London, and a few days later was in Sweden talking about randomness in rule 30 and its potential significance.

A month or so later, writing an article for *Scientific American*—nominally on the subject of software in science and mathematics—led me to think more carefully about basic issues of computation and modelling, and to describe for the first time the idea of computational irreducibility (see page 737). In the fall of 1984 I began to investigate some of the implications of what I had discovered about cellular automata for foundational questions in science. And by early 1985 I had written what I consider to be my two most fundamental (if excessively short) papers from the period: one on undecidability and intractability in theoretical physics, and the other on intrinsic randomness generation and the origins of randomness in physical systems.

In the early summer of 1985 I was doing consulting at a startup company called Thinking Machines Corporation, which had developed a massively parallel computer called the Connection Machine that was fairly well suited to cellular automaton simulation. Partly as an application for this computer I then ended up making a detailed study of rule 30 and its randomness—among other things proposing it as a practical random sequence generator and cryptosystem.

I had always thought that cellular automata could be a way to get at foundational questions in thermodynamics and hydrodynamics. And in mid-1985, partly in an attempt to find uses for the Connection Machine, I devised a practical scheme for doing fluid mechanics with cellular automata (see page 378). Then over the course of that winter and the following spring I analyzed the scheme and worked out its correspondence to the traditional continuum approach.

By 1986, however, I felt that I had answered at least the first round of obvious questions about cellular automata, and it increasingly seemed that it would not be easier to go further with the computational tools available. In June 1986 I organized one last conference on cellular automata—then in August 1986 essentially left the field to begin the development of *Mathematica*.

Over the years, I have come back to look at cellular automata again and again, and every time I have been amazed and delighted by the richness of the phenomena they exhibit. As I argue in this book, a vast range of systems must in the end show the same basic phenomena. But cellular automata—and especially 1D ones—make the phenomena particularly clear, which is why even after investigating all sorts of other systems 1D cellular automata are still the most common examples that I use in this book.

▪ **My papers.** The primary papers that I published about cellular automata and other issues related to this book were (the dates indicate when I finished my work on each paper; the papers were actually published 6–12 months later):

- "Statistical mechanics of cellular automata" (June 1982) (introducing 1D cellular automata and studying many of their properties)
- "Algebraic properties of cellular automata" (with Olivier Martin and Andrew Odlyzko) (February 1983) (analyzing additive cellular automata such as rule 90)
- "Universality and complexity in cellular automata" (April 1983) (classifying cellular automaton behavior)
- "Computation theory of cellular automata" (November 1983) (characterizing behavior using formal language theory)

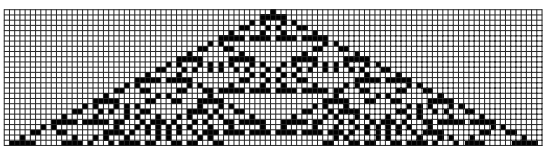
- "Two-dimensional cellular automata" (with Norman Packard) (October 1984) (extending results to two dimensions)
- "Undecidability and intractability in theoretical physics" (October 1984) (introducing computational irreducibility)
- "Origins of randomness in physical systems" (February 1985) (introducing intrinsic randomness generation)
- "Random sequence generation by cellular automata" (July 1985) (a detailed study of rule 30)
- "Thermodynamics and hydrodynamics of cellular automata" (with James Salem) (November 1985) (continuum behavior from cellular automata)
- "Approaches to complexity engineering" (December 1985) (finding systems that achieve specified goals)
- "Cellular automaton fluids: Basic theory" (March 1986) (deriving the Navier-Stokes equations from cellular automata)

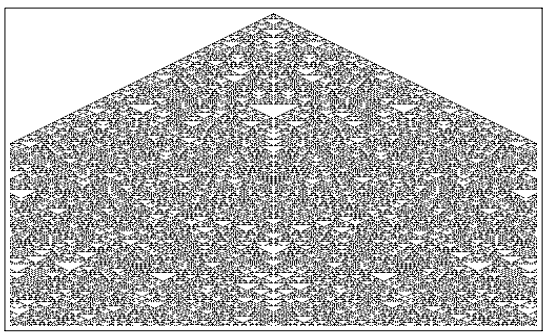
The ideas in the first five and the very last of these papers have been reasonably well absorbed over the past fifteen or so years. But those in the other five have not, and indeed seem to require the whole development of this book to be able to present in an appropriate way.

Other significant publications of mine providing relevant summaries were (the dates here are for actual publication—sometimes close to writing, but sometimes long delayed):

- "Computers in science and mathematics" (September 1984) (*Scientific American* article about foundations of the computational approach to science and mathematics)
- "Cellular automata as models of complexity" (October 1984) (*Nature* article introducing cellular automata)
- "Geometry of binomial coefficients" (November 1984) (additive cellular automata and nested patterns)
- "Twenty problems in the theory of cellular automata" (1985) (a list of unsolved problems to attack—most now finally resolved in this book)
- "Tables of cellular automaton properties" (June 1986) (features of elementary cellular automata)
- "Cryptography with cellular automata" (1986) (using rule 30 as a cryptosystem)
- "Complex systems theory" (1988) (1984 speech suggesting the research direction for the new Santa Fe Institute)

■ **Code 10.** Rule 30 is by many measures the simplest cellular automaton that generates randomness from a single black initial cell. But there are other simple examples—that historically I noticed slightly earlier than rule 30, though did not study—that occur in $k = 2$, $r = 2$ totalistic rules. And indeed among the 64 such rules, 13 show randomness. An example shown below is code 10, which specifies that if 1 or 3 cells out of 5 are black then the next cell is black; otherwise it is white.





The World of Simple Programs

More Cellular Automata

▪ **Page 53 · Numbering scheme.** I introduced the numbering scheme used here in the 1983 paper where I first discussed one-dimensional cellular automata (see page 881). I termed two-color nearest-neighbor cellular automata "elementary" to reflect the idea that their rules are as simple as possible.

▪ **Page 55 · Rule equivalences.** The table below gives basic equivalences between elementary cellular automaton rules. In each block the second entry is the rule obtained by interchanging black and white, the third entry is the rule obtained by interchanging left and right, and the fourth entry the rule obtained by applying both operations. (The smallest rule number is given in boldface.) For a rule with number *n* the two operations correspond respectively to computing *1 – Reverse[list]* and *list[[{1, 5, 3, 7, 2, 6, 4, 8}]]* with *list = IntegerDigits[n, 2, 8]*.

▪ **Special rules.** Rule 51: complement; rule 170: left shift; rule 204: identity; rule 240: right shift. These rules only ever depend on one cell in each neighborhood.

▪ **Rule expressions.** The table on the next page gives Boolean expressions for each of the elementary rules. The expressions

0	255	**0**	255	**32**	251	**32**	251	64	253	**8**	239	96	249	**40**	235	**128**	254	**128**	254	**160**	250	**160**	250	192	252	**136**	238	224	248	**168**	234
1	127	**1**	127	**33**	123	**33**	123	65	125	**9**	111	97	121	**41**	107	129	**126**	129	**126**	161	**122**	161	**122**	193	124	137	**110**	225	120	169	**106**
2	191	16	247	**34**	187	48	243	66	189	**24**	231	98	185	**56**	227	**130**	190	144	246	**162**	186	176	242	194	188	**152**	230	226	**184**	**184**	226
3	63	17	119	**35**	59	49	115	67	61	**25**	103	99	**57**	**57**	99	131	**62**	145	118	163	**58**	177	114	195	**60**	153	102	227	**56**	185	98
4	223	**4**	223	**36**	219	**36**	219	68	221	**12**	207	100	217	**44**	203	**132**	222	**132**	222	**164**	218	**164**	218	196	220	**140**	206	228	216	**172**	202
5	95	**5**	95	**37**	91	**37**	91	69	93	**13**	79	101	89	**45**	75	133	**94**	133	**94**	165	**90**	165	**90**	197	92	141	**78**	229	88	173	**74**
6	159	20	215	**38**	155	52	211	70	157	**28**	199	102	153	**60**	195	**134**	158	148	214	166	**154**	180	210	198	**156**	**156**	198	230	**152**	188	194
7	31	21	87	39	**27**	53	83	71	**29**	**29**	71	103	**25**	61	67	135	**30**	149	86	167	**26**	181	82	199	**28**	157	70	231	**24**	189	66
8	239	64	253	**40**	235	96	249	**72**	237	**72**	237	**104**	233	**104**	233	**136**	238	192	252	**168**	234	224	248	**200**	236	**200**	236	**232**	**232**	**232**	**232**
9	111	65	125	**41**	107	97	121	**73**	109	**73**	109	**105**	**105**	**105**	**105**	137	**110**	193	124	169	**106**	225	120	201	**108**	201	**108**	233	**104**	233	**104**
10	175	80	245	**42**	171	112	241	**74**	173	88	229	**106**	169	120	225	**138**	174	208	244	**170**	**170**	240	240	202	**172**	216	228	234	**168**	248	224
11	47	81	117	**43**	**43**	113	113	75	**45**	89	101	107	**41**	121	97	139	**46**	209	116	171	**42**	241	112	203	**44**	217	100	235	**40**	249	96
12	207	68	221	**44**	203	100	217	**76**	205	**76**	205	**108**	201	**108**	201	**140**	206	196	220	**172**	202	228	216	**204**	**204**	**204**	**204**	236	**200**	236	**200**
13	79	69	93	**45**	75	101	89	**77**	**77**	**77**	**77**	109	**73**	109	**73**	141	**78**	197	92	173	**74**	229	88	205	**76**	205	**76**	237	**72**	237	**72**
14	143	84	213	**46**	139	116	209	**78**	141	92	197	**110**	137	124	193	**142**	**142**	212	212	174	**138**	244	208	206	**140**	220	196	238	**136**	252	192
15	**15**	85	85	47	**11**	117	81	79	**13**	93	69	111	**9**	125	65	143	**14**	213	84	175	**10**	245	80	207	**12**	221	68	239	**8**	253	64
16	247	**2**	191	48	243	**34**	187	80	245	**10**	175	112	241	**42**	171	144	246	**130**	190	176	242	**162**	186	208	244	**138**	174	240	240	**170**	**170**
17	119	**3**	63	49	115	**35**	59	81	117	**11**	47	113	113	**43**	**43**	145	118	131	**62**	177	114	163	**58**	209	116	139	**46**	241	112	171	**42**
18	183	**18**	183	**50**	179	**50**	179	82	181	**26**	167	114	177	**58**	163	**146**	182	**146**	182	**178**	**178**	**178**	**178**	210	180	**154**	166	242	176	186	**162**
19	55	**19**	55	**51**	**51**	**51**	**51**	83	53	**27**	39	115	49	59	**35**	147	**54**	147	**54**	179	**50**	179	**50**	211	52	155	**38**	243	48	187	**34**
20	215	**6**	159	52	211	**38**	155	84	213	**14**	143	116	209	**46**	139	148	214	**134**	158	180	210	166	**154**	212	212	**142**	**142**	244	208	174	**138**
21	87	**7**	31	53	83	39	**27**	85	85	**15**	**15**	117	81	47	**11**	149	86	135	**30**	181	82	167	**26**	213	84	143	**14**	245	80	175	**10**
22	151	**22**	151	**54**	147	**54**	147	86	149	**30**	135	118	145	**62**	131	**150**	**150**	**150**	**150**	182	**146**	182	**146**	214	148	158	**134**	246	144	190	**130**
23	**23**	**23**	**23**	55	**19**	55	**19**	87	21	31	**7**	119	17	63	**3**	151	**22**	151	**22**	183	**18**	183	**18**	215	20	159	**6**	247	16	191	**2**
24	231	66	189	**56**	227	98	185	88	229	**74**	173	120	225	**106**	169	**152**	230	194	188	**184**	226	226	**184**	216	228	202	**172**	248	224	234	**168**
25	103	67	61	**57**	99	99	**57**	89	101	75	**45**	121	97	107	**41**	153	102	195	**60**	185	98	227	**56**	217	100	203	**44**	249	96	235	**40**
26	167	82	181	**58**	163	114	177	**90**	165	**90**	165	**122**	161	**122**	161	**154**	166	210	180	186	**162**	242	176	218	**164**	218	**164**	250	**160**	250	**160**
27	39	83	53	59	**35**	115	49	91	**37**	91	**37**	123	**33**	123	**33**	155	**38**	211	52	187	**34**	243	48	219	**36**	219	**36**	251	**32**	251	**32**
28	199	70	157	**60**	195	102	153	92	197	**78**	141	124	193	**110**	137	**156**	198	198	**156**	188	194	230	**152**	220	196	206	**140**	252	192	238	**136**
29	71	71	**29**	61	67	103	**25**	93	69	79	**13**	125	65	111	**9**	157	70	199	**28**	189	66	231	**24**	221	68	207	**12**	253	64	239	**8**
30	135	86	149	**62**	131	118	145	**94**	133	**94**	133	**126**	129	**126**	129	158	**134**	214	148	190	**130**	246	144	222	**132**	222	**132**	254	**128**	254	**128**
31	**7**	87	21	63	**3**	119	17	95	**5**	95	**5**	127	**1**	127	**1**	159	**6**	215	20	191	**2**	247	16	223	**4**	223	**4**	255	**0**	255	**0**

rule 0 : 0
rule 1 : $\neg(p \vee q \vee r)$
rule 2 : $(\neg p) \wedge (\neg q) \wedge r$
rule 3 : $\neg(p \vee q)$
rule 4 : $(\neg(p \vee r)) \wedge q$
rule 5 : $\neg(p \vee r)$
rule 6 : $(\neg p) \wedge (q \veebar r)$
rule 7 : $\neg(p \vee (q \wedge r))$
rule 8 : $(\neg p) \wedge q \wedge r$
rule 9 : $\neg(p \vee (q \veebar r))$
rule 10 : $(\neg p) \wedge r$
rule 11 : $p \veebar (p \vee (\neg q) \vee r)$
rule 12 : $(p \wedge q) \veebar q$
rule 13 : $p \veebar (p \vee q \vee (\neg r))$
rule 14 : $p \veebar (p \vee q \vee r)$
rule 15 : $\neg p$
rule 16 : $p \wedge (\neg q) \wedge (\neg r)$
rule 17 : $\neg(q \vee r)$
rule 18 : $(p \veebar q \veebar r) \wedge (\neg q)$
rule 19 : $\neg((p \wedge r) \vee q)$
rule 20 : $(p \veebar q) \wedge (\neg r)$
rule 21 : $\neg((p \wedge q) \vee r)$
rule 22 : $p \veebar (p \wedge q \wedge r) \veebar q \veebar r$
rule 23 : $p \veebar ((p \veebar (\neg q)) \vee (q \veebar r))$
rule 24 : $(p \veebar q) \wedge (p \veebar r)$
rule 25 : $(p \wedge q \wedge r) \veebar q \veebar (\neg r)$
rule 26 : $p \veebar ((p \wedge q) \vee r)$
rule 27 : $p \veebar ((p \veebar (\neg q)) \vee r)$
rule 28 : $p \veebar ((p \wedge r) \vee q)$
rule 29 : $p \veebar ((p \veebar (\neg r)) \vee q)$
rule 30 : $p \veebar (q \vee r)$
rule 31 : $\neg(p \wedge (q \vee r))$
rule 32 : $p \wedge (\neg q) \wedge r$
rule 33 : $\neg((p \veebar q \veebar r) \vee q)$
rule 34 : $(\neg q) \wedge r$
rule 35 : $((\neg p) \vee q \vee r) \veebar q$
rule 36 : $(p \veebar q) \wedge (q \veebar r)$
rule 37 : $p \veebar (p \wedge q \wedge r) \veebar (\neg r)$
rule 38 : $((p \wedge q) \vee r) \veebar q$
rule 39 : $((p \veebar (\neg q)) \vee r) \veebar q$
rule 40 : $(p \veebar q) \wedge r$
rule 41 : $\neg((p \wedge q) \vee (p \veebar q \veebar r))$
rule 42 : $(p \wedge q \wedge r) \veebar r$
rule 43 : $p \veebar ((p \veebar r) \vee (p \veebar (\neg q)))$
rule 44 : $(p \wedge (q \vee r)) \veebar q$
rule 45 : $p \veebar (q \vee (\neg r))$
rule 46 : $(p \wedge q) \veebar (q \vee r)$
rule 47 : $(\neg p) \vee ((\neg q) \wedge r)$
rule 48 : $p \wedge (\neg q)$
rule 49 : $(p \vee q \vee (\neg r)) \veebar q$
rule 50 : $(p \vee q \vee r) \veebar q$
rule 51 : $\neg q$
rule 52 : $(p \vee (q \wedge r)) \veebar q$
rule 53 : $(p \vee (q \veebar (\neg r))) \veebar q$
rule 54 : $(p \vee r) \veebar q$
rule 55 : $\neg((p \vee r) \wedge q)$
rule 56 : $p \veebar ((p \vee r) \wedge q)$
rule 57 : $(p \vee (\neg r)) \veebar q$
rule 58 : $(p \vee (q \veebar r)) \veebar q$
rule 59 : $((\neg p) \wedge r) \vee (\neg q)$
rule 60 : $p \veebar q$
rule 61 : $p \veebar (p \vee q \vee r) \veebar (\neg q)$
rule 62 : $(p \wedge q) \veebar (p \vee q \vee r)$
rule 63 : $\neg(p \wedge q)$
rule 64 : $p \wedge q \wedge (\neg r)$
rule 65 : $\neg((p \veebar q) \vee r)$
rule 66 : $(p \veebar r) \wedge (q \veebar r)$
rule 67 : $p \veebar (p \wedge q \wedge r) \veebar (\neg q)$
rule 68 : $q \wedge (\neg r)$
rule 69 : $((\neg p) \vee q \vee r) \veebar r$
rule 70 : $((p \wedge r) \vee q) \veebar r$
rule 71 : $((p \veebar (\neg r)) \vee q) \veebar r$
rule 72 : $(p \wedge q) \veebar (q \wedge r)$
rule 73 : $\neg((p \wedge r) \vee (p \veebar q \veebar r))$
rule 74 : $(p \wedge (q \vee r)) \veebar r$
rule 75 : $p \veebar ((\neg q) \vee r)$
rule 76 : $(p \wedge q \wedge r) \veebar q$
rule 77 : $p \veebar ((p \veebar q) \vee (p \veebar (\neg r)))$
rule 78 : $p \veebar ((p \veebar q) \vee r)$
rule 79 : $(\neg p) \vee (q \wedge (\neg r))$
rule 80 : $p \wedge (\neg r)$
rule 81 : $(p \vee (\neg q) \vee r) \veebar r$
rule 82 : $(p \vee (q \wedge r)) \veebar r$
rule 83 : $(p \vee (q \veebar (\neg r))) \veebar r$
rule 84 : $(p \vee q \vee r) \veebar r$
rule 85 : $\neg r$
rule 86 : $(p \vee q) \veebar r$
rule 87 : $\neg((p \vee q) \wedge r)$
rule 88 : $p \veebar ((p \vee q) \wedge r)$
rule 89 : $(p \vee (\neg q)) \veebar r$
rule 90 : $p \veebar r$
rule 91 : $p \veebar (\neg(p \vee q \vee r)) \veebar r$
rule 92 : $(p \vee (q \veebar r)) \veebar r$
rule 93 : $\neg((p \vee (\neg q)) \wedge r)$
rule 94 : $(p \wedge r) \veebar (p \vee q \vee r)$
rule 95 : $\neg(p \wedge r)$
rule 96 : $p \wedge (q \veebar r)$
rule 97 : $\neg((p \veebar q \veebar r) \vee (q \wedge r))$
rule 98 : $((p \vee r) \wedge q) \veebar r$
rule 99 : $((\neg p) \vee r) \veebar q$
rule 100 : $((p \vee q) \wedge r) \veebar q$
rule 101 : $p \veebar (p \wedge q) \veebar (\neg r)$
rule 102 : $q \veebar r$
rule 103 : $(\neg(p \vee q \vee r)) \veebar q \veebar r$
rule 104 : $p \veebar (p \vee q \vee r) \veebar q \veebar r$
rule 105 : $p \veebar q \veebar (\neg r)$
rule 106 : $(p \wedge q) \veebar r$
rule 107 : $p \veebar (p \vee q \vee (\neg r)) \veebar q \veebar r$
rule 108 : $(p \wedge r) \veebar q$
rule 109 : $p \veebar (p \vee (\neg q) \vee r) \veebar q \veebar r$
rule 110 : $((\neg p) \wedge q \wedge r) \veebar q \veebar r$
rule 111 : $(\neg p) \vee (q \veebar r)$
rule 112 : $p \veebar (p \wedge q \wedge r)$
rule 113 : $p \veebar (\neg((p \veebar q) \vee (p \veebar r)))$
rule 114 : $((p \veebar q) \vee r) \veebar q$
rule 115 : $(p \wedge (\neg r)) \vee (\neg q)$
rule 116 : $(p \vee q) \veebar (q \wedge r)$
rule 117 : $(p \wedge (\neg q)) \vee (\neg r)$
rule 118 : $(p \vee q \vee r) \veebar (q \wedge r)$
rule 119 : $\neg(q \wedge r)$
rule 120 : $p \veebar (q \wedge r)$
rule 121 : $p \veebar ((\neg p) \vee q \vee r) \veebar q \veebar r$
rule 122 : $p \veebar (p \wedge (\neg q) \wedge r) \veebar r$
rule 123 : $\neg((p \veebar q \veebar r) \wedge q)$
rule 124 : $p \veebar (p \wedge q \wedge (\neg r)) \veebar q$
rule 125 : $(p \veebar q) \vee (\neg r)$
rule 126 : $(p \veebar q) \vee (p \veebar r)$
rule 127 : $\neg(p \wedge q \wedge r)$
rule 128 : $p \wedge q \wedge r$
rule 129 : $\neg((p \veebar q) \vee (p \veebar r))$
rule 130 : $(p \veebar q \veebar r) \wedge r$
rule 131 : $p \veebar (p \wedge q \wedge (\neg r)) \veebar (\neg q)$
rule 132 : $(p \veebar q \veebar r) \wedge q$
rule 133 : $p \veebar (p \wedge (\neg q) \wedge r) \veebar (\neg r)$
rule 134 : $(p \wedge (q \vee r)) \veebar q \veebar r$
rule 135 : $(\neg p) \veebar (q \wedge r)$
rule 136 : $q \wedge r$
rule 137 : $((\neg p) \vee q \vee r) \veebar q \veebar r$
rule 138 : $(p \wedge (\neg q) \wedge r) \veebar r$
rule 139 : $\neg((p \vee q) \veebar (q \wedge r))$
rule 140 : $((\neg p) \vee r) \wedge q$
rule 141 : $p \veebar ((p \veebar q) \vee (\neg r))$
rule 142 : $p \veebar ((p \veebar q) \vee (p \veebar r))$
rule 143 : $(\neg p) \vee (q \wedge r)$
rule 144 : $p \wedge (p \veebar q \veebar r)$
rule 145 : $((\neg p) \wedge q \wedge r) \veebar q \veebar (\neg r)$
rule 146 : $p \veebar ((p \vee r) \wedge q) \veebar r$
rule 147 : $(p \wedge r) \veebar (\neg q)$
rule 148 : $p \veebar ((p \vee q) \wedge r) \veebar q$
rule 149 : $(p \wedge q) \veebar (\neg r)$
rule 150 : $p \veebar q \veebar r$
rule 151 : $p \veebar (\neg(p \vee q \vee r)) \veebar q \veebar r$
rule 152 : $(p \vee q \vee r) \veebar q \veebar r$
rule 153 : $q \veebar (\neg r)$
rule 154 : $p \veebar (p \wedge q) \veebar r$
rule 155 : $(p \vee q \vee (\neg r)) \veebar q \veebar r$
rule 156 : $p \veebar (p \wedge r) \veebar q$
rule 157 : $(p \vee (\neg q) \vee r) \veebar q \veebar r$
rule 158 : $(p \veebar q \veebar r) \vee (q \wedge r)$
rule 159 : $\neg(p \wedge (q \veebar r))$
rule 160 : $p \wedge r$
rule 161 : $p \veebar (p \vee (\neg q) \vee r) \veebar r$
rule 162 : $(p \vee (\neg q)) \wedge r$
rule 163 : $((\neg p) \vee (q \veebar r)) \veebar q$
rule 164 : $p \veebar (p \vee q \vee r) \veebar r$
rule 165 : $p \veebar (\neg r)$
rule 166 : $(p \wedge q) \veebar q \veebar r$
rule 167 : $p \veebar (p \vee q \vee (\neg r)) \veebar r$
rule 168 : $(p \vee q) \wedge r$
rule 169 : $(\neg(p \vee q)) \veebar r$
rule 170 : r
rule 171 : $(\neg(p \vee q)) \vee r$
rule 172 : $(p \wedge (q \veebar r)) \veebar q$
rule 173 : $(p \veebar (\neg r)) \vee (q \wedge r)$
rule 174 : $((p \wedge q) \veebar q) \vee r$
rule 175 : $(\neg p) \vee r$
rule 176 : $p \wedge ((\neg q) \vee r)$
rule 177 : $p \veebar (\neg((p \veebar q) \vee r))$
rule 178 : $((p \veebar q) \vee (p \veebar r)) \veebar q$
rule 179 : $(p \wedge r) \vee (\neg q)$
rule 180 : $p \veebar q \veebar (q \wedge r)$
rule 181 : $p \veebar ((\neg p) \vee q \vee r) \veebar r$
rule 182 : $(p \wedge r) \vee (p \veebar q \veebar r)$
rule 183 : $(p \veebar q \veebar r) \vee (\neg q)$
rule 184 : $p \veebar (p \wedge q) \veebar (q \wedge r)$
rule 185 : $(p \wedge r) \vee (q \veebar (\neg r))$
rule 186 : $(p \wedge (\neg q)) \vee r$
rule 187 : $(\neg q) \vee r$
rule 188 : $p \veebar (p \wedge q \wedge r) \veebar q$
rule 189 : $(p \veebar q) \vee (p \veebar (\neg r))$
rule 190 : $(p \veebar q) \vee r$
rule 191 : $(\neg p) \vee (\neg q) \vee r$
rule 192 : $p \wedge q$
rule 193 : $p \veebar (p \vee q \vee (\neg r)) \veebar q$
rule 194 : $p \veebar (p \vee q \vee r) \veebar q$
rule 195 : $p \veebar (\neg q)$
rule 196 : $(p \vee (\neg r)) \wedge q$
rule 197 : $(\neg(p \vee (q \veebar r))) \veebar q$
rule 198 : $(p \wedge r) \veebar q \veebar r$
rule 199 : $p \veebar (p \vee (\neg q) \vee r) \veebar q$
rule 200 : $(p \vee r) \wedge q$
rule 201 : $(\neg(p \vee r)) \veebar q$
rule 202 : $(p \wedge (q \veebar r)) \veebar r$
rule 203 : $(p \veebar (\neg q)) \vee (q \wedge r)$
rule 204 : q
rule 205 : $(\neg(p \vee r)) \vee q$
rule 206 : $((\neg p) \wedge r) \vee q$
rule 207 : $\neg(p \wedge (\neg q))$
rule 208 : $p \wedge (q \vee (\neg r))$
rule 209 : $\neg((p \wedge q) \veebar (q \vee r))$
rule 210 : $p \veebar (q \wedge r) \veebar r$
rule 211 : $p \veebar ((\neg p) \vee q \vee r) \veebar q$
rule 212 : $((p \veebar q) \vee (p \veebar r)) \veebar r$
rule 213 : $(p \wedge q) \vee (\neg r)$
rule 214 : $(p \wedge q) \vee (p \veebar q \veebar r)$
rule 215 : $\neg((p \veebar q) \wedge r)$
rule 216 : $p \veebar ((p \veebar q) \wedge r)$
rule 217 : $(p \wedge q) \vee (q \veebar (\neg r))$
rule 218 : $p \veebar (p \wedge q \wedge r) \veebar r$
rule 219 : $(p \veebar r) \vee (p \veebar (\neg q))$
rule 220 : $(p \wedge (\neg r)) \vee q$
rule 221 : $q \vee (\neg r)$
rule 222 : $(p \veebar q \veebar r) \vee q$
rule 223 : $\neg(p \wedge (\neg q) \wedge r)$
rule 224 : $p \wedge (q \vee r)$
rule 225 : $p \veebar (\neg(q \vee r))$
rule 226 : $(p \wedge q) \veebar (q \wedge r) \veebar r$
rule 227 : $(p \wedge r) \vee (p \veebar (\neg q))$
rule 228 : $((p \veebar q) \wedge r) \veebar q$
rule 229 : $(p \wedge q) \vee (p \veebar (\neg r))$
rule 230 : $(p \wedge q \wedge r) \veebar q \veebar r$
rule 231 : $(p \veebar (\neg q)) \vee (q \veebar r)$
rule 232 : $(p \wedge q) \vee ((p \vee q) \wedge r)$
rule 233 : $p \veebar (p \wedge q \wedge r) \veebar q \veebar (\neg r)$
rule 234 : $(p \wedge q) \vee r$
rule 235 : $(p \veebar (\neg q)) \vee r$
rule 236 : $(p \wedge r) \vee q$
rule 237 : $(p \veebar (\neg r)) \vee q$
rule 238 : $q \vee r$
rule 239 : $(\neg p) \vee q \vee r$
rule 240 : p
rule 241 : $p \vee (\neg(q \vee r))$
rule 242 : $p \vee ((\neg q) \wedge r)$
rule 243 : $p \vee (\neg q)$
rule 244 : $p \vee (q \wedge (\neg r))$
rule 245 : $p \vee (\neg r)$
rule 246 : $p \vee (q \veebar r)$
rule 247 : $p \vee (\neg q) \vee (\neg r)$
rule 248 : $p \vee (q \wedge r)$
rule 249 : $p \vee (q \veebar (\neg r))$
rule 250 : $p \vee r$
rule 251 : $p \vee (\neg q) \vee r$
rule 252 : $p \vee q$
rule 253 : $p \vee q \vee (\neg r)$
rule 254 : $p \vee q \vee r$
rule 255 : 1

use the minimum possible number of operators; when there are several equivalent forms, I give the most uniform and symmetrical one. Note that ⊻ stands for *Xor*.

■ **Rule orderings.** The fact that successive rules often show very different behavior does not appear to be affected by using alternative orderings such as Gray code (see page 901.)

■ **Page 58 · Algebraic forms.** The rules here can be expressed in algebraic terms (see page 869) as follows:

- Rule 22: *Mod[p + q + r + p q r, 2]*
- Rule 60: *Mod[p + q, 2]*
- Rule 105: *Mod[1 + p + q + r, 2]*
- Rule 129: *Mod[1 + p + q + r + p q + q r + p r, 2]*
- Rule 150: *Mod[p + q + r, 2]*
- Rule 225: *Mod[1 + p + q + r + q r, 2]*

Note that rules 60, 105 and 150 are additive, like rule 90.

■ **Rule 150.** This rule can be viewed as an analog of rule 90 in which the values of three cells, rather than two, are added modulo 2. Corresponding to the result on page 870 for rule 90, the number of black cells at row t in the pattern from rule 150 is given by

```
Apply[Times, Map[(2^#+2 - (-1)^#+2)/3 &,
    Cases[Split[IntegerDigits[t, 2]], k : {(1)..} :> Length[k]]]]
```

There are a total of 2^m *Fibonacci[m + 2]* black cells in the pattern obtained up to step 2^m, implying fractal dimension $Log[2, 1 + \sqrt{5}]$. (See also page 956.)

The value at step t in the column immediately adjacent to the center is the nested sequence discussed on page 892 and given by *Mod[IntegerExponent[t, 2], 2]*. The cell at position n on row t turns out to be given by *Mod[GegenbauerC[n, -t, -1/2], 2]*, as discussed on page 612.

■ **Rule 225.** The width of the pattern after t steps varies between $Sqrt[3/2]\sqrt{t}$ (achieved when $t = 3 \times 2^{2n+1}$) and $Sqrt[9/2]\sqrt{t}$ (achieved when $t = 2^{2n+1}$). The pattern scales differently in the horizontal and vertical direction, corresponding to fractal dimensions *Log[2, 5]* and *Log[4, 5]* respectively. Note that with more complicated initial conditions rule 225 often no longer yields a regular nested pattern, as shown on page 951. The resulting patterns typically grow at a roughly constant average rate.

■ **Rule 22.** With more complicated initial conditions the pattern is often no longer nested, as shown on page 263.

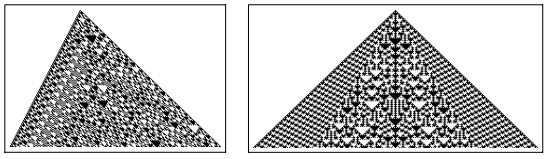
■ **Page 59 · Algebraic forms.** The rules here can be expressed in algebraic terms (see page 869) as follows:

- Rule 30: *Mod[p + q + r + q r, 2]*
- Rule 45: *Mod[1 + p + r + q r, 2]*
- Rule 73: *Mod[1 + p + q + r + p r + p q r, 2]*

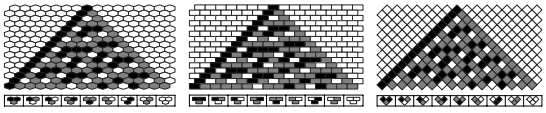
■ **Rule 45.** The center column of the pattern appears for practical purposes random, just as in rule 30. The left edge of the pattern moves 1 cell every 2 steps; the boundary between repetition and randomness moves on average 0.17 cells per step.

■ **Rule 73.** The pattern has a few definite regularities. The center column of cells is repetitive, alternating between black and white on successive steps. And in all cases black cells appear only in blocks that are an odd number of cells wide. (Any block in rule 73 consisting of an even number of black cells will evolve to a structure that remains fixed forever, as mentioned on page 954.) The more complicated central region of the pattern grows 4 cells every 7 steps; the outer region consists of blocks that are 12 cells wide and repeat every 3 steps.

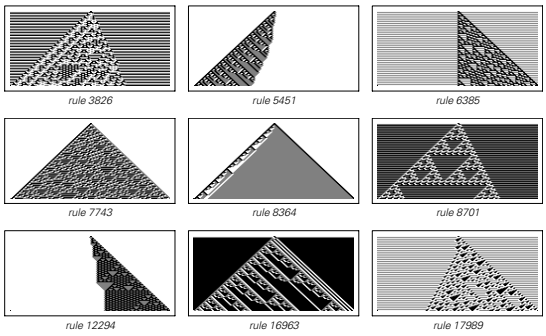
■ **Alternating colors.** The pictures below show rules 45 and 73 with the colors of cells on alternate steps reversed.



■ **Two-cell neighborhoods.** By having cells on successive steps be arranged like hexagons or staggered bricks, as in the pictures below, one can set up cellular automata in which the new color of each cell depends on the previous colors of two rather than three neighboring cells.



With k possible colors for each cell, there are a total of k^{k^2} possible rules of this type, each specified by a k^2-digit number in base k (7743 for the rule shown above). For $k = 2$, there are 16 possible rules, and the most complicated pattern obtained is nested like the rule 90 elementary cellular automaton. With $k = 3$, there are 19,683 possible rules, 1734 of which are fundamentally inequivalent, and many more complicated patterns are seen, as in the pictures at the top of the next page.



With *rule* given by *IntegerDigits[num, k, k^2]* a single step of evolution can be implemented as

```
CAStep[{k_, rule_}, a_List] := rule[[k^2 - RotateLeft[a] - k a]]
```

▪ **Page 60 · Numbers of rules.** Allowing k possible colors for each cell and considering r neighbors on each side, there are $k^{k^{2r+1}}$ possible cellular automaton rules in all, of which $k^{1/2\,k^{r+1}\,(1+k^r)}$ are symmetric, and $k^{1+(k-1)\,(2r+1)}$ are totalistic. (For $k = 2$, $r = 1$ there are therefore 256 possible rules altogether, of which 16 are totalistic. For $k = 2$, $r = 2$ there are 4,294,967,296 rules in all, of which 64 are totalistic. And for $k = 3$, $r = 1$ there are 7,625,597,484,987 rules in all, with 2187 totalistic ones.) Note that for $k > 2$, a particular rule will in general be totalistic only for a specific assignment of values to colors. I first introduced totalistic rules in 1983.

▪ **Implementation of general cellular automata.** With k colors and r neighbors on each side, a single step in the evolution of a general cellular automaton is given by

```
CAStep[CARule[rule_List, k_, r_], a_List] :=
  rule[[-1 - ListConvolve[k^Range[0, 2r], a, r + 1]]]
```

where *rule* is obtained from a rule number *num* by *IntegerDigits[num, k, k^(2r+1)]*. (See also page 927.)

▪ **Implementation of totalistic cellular automata.** To handle totalistic rules that involve k colors and nearest neighbors, one can add the definition

```
CAStep[TotalisticCARule[rule_List, 1], a_List] :=
  rule[[-1 - (RotateLeft[a] + a + RotateRight[a])]]
```

to what was given on page 867. The following definition also handles the more general case of r neighbors:

```
CAStep[TotalisticCARule[rule_List, r_Integer], a_List] :=
  rule[[-1 - Sum[RotateLeft[a, i], {i, -r, r}]]]
```

One can generate the representation of totalistic rules used by these functions from code numbers using

```
ToTotalisticCARule[num_Integer, k_Integer, r_Integer] :=
 TotalisticCARule[IntegerDigits[num, k, 1 + (k - 1) (2r + 1)], r]
```

▪ **Common framework.** The *Mathematica* built-in function *CellularAutomaton* discussed on page 867 handles general and totalistic rules in the same framework by using *ListConvolve[w, a, r + 1]* and taking the weights w to be respectively *k^Table[i - 1, {i, 2r + 1}]* and *Table[1, {2r + 1}]*.

▪ **Page 63 · Mod 3 rule.** Code 420 is an example of an additive rule, and yields a pattern corresponding to Pascal's triangle modulo 3, as discussed on page 870.

▪ **Compositions of cellular automata.** One way to construct more complicated rules is from compositions of simpler rules. One can, for example, consider each step applying first one elementary cellular automaton rule, then another. The result is in effect a $k = 2$, $r = 2$ rule. Usually the order in which the two elementary rules are applied will matter, and the overall behavior obtained will have no simple relationship to that of either of the individual rules. (See also page 956.)

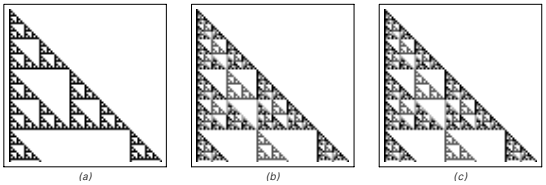
▪ **Rules based on algebraic systems.** If the values of cells are taken to be elements of some finite algebraic system, then one can set up a cellular automaton with rule

```
a[t_, i_] := f[a[t - 1, i - 1], a[t - 1, i]]
```

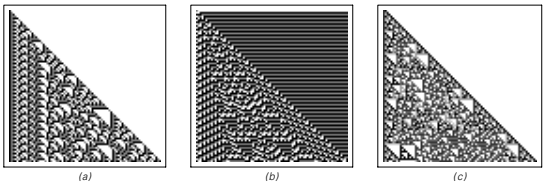
where f is the analog of multiplication for the system (see also page 1094). The pattern obtained after t steps is then given by

```
NestList[f[RotateRight[#], #] &, init, t]
```

The pictures below show results with f being *Times*, and cells having values (a) *{1, -1}*, (b) the unit complex numbers *{1, i, -1, -i}*, (c) the unit quaternions.



In general, with n elements f can be specified by an $n \times n$ "multiplication table". For $n = 2$, the patterns obtained are at most nested. Pictures (a) and (b) below however correspond to the $n = 3$ multiplication tables *{{1, 1, 3}, {3, 3, 2}, {2, 2, 1}}* and *{{3, 1, 3}, {1, 3, 1}, {3, 1, 2}}*. Note that for (b) the table is symmetric, corresponding to a commutative multiplication operation.



If f is associative (flat), so that *f[f[i, j], k] == f[i, f[j, k]]*, then the algebraic system is known as a semigroup. (See also

page 805.) With a single cell seed, no pattern more complicated than nested can be obtained in such a system. And with any seed, it appears to require a semigroup with at least six elements to obtain a more complicated pattern.

If *f* has an identity element, so that *f[1, i] == i* for all *i*, and has inverses, so that *f[i, j] == 1* for some *j*, then the system is a group. (See page 945.) If the group is Abelian, so that *f[i, j] == f[j, i]*, then only nested patterns are ever produced (see page 955). But it turns out that the very simplest possible non-Abelian group yields the pattern in (c) above. The group used is S_3, which has six elements and multiplication table

```
{{1, 2, 3, 4, 5, 6}, {2, 1, 5, 6, 3, 4}, {3, 4, 1, 2, 6, 5},
 {4, 3, 6, 5, 1, 2}, {5, 6, 2, 1, 4, 3}, {6, 5, 4, 3, 2, 1}}
```

The initial condition contains *{5, 6}* surrounded by *1*'s.

Mobile Automata

■ **Implementation.** The state of a mobile automaton at a particular step can conveniently be represented by a pair *{list, n}*, where *list* gives the values of the cells, and *n* specifies the position of the active cell (the value of the active cell is thus *list[[n]]*). Then, for example, the rule for the mobile automaton shown on page 71 can be given as

```
{{1, 1, 1} → {0, 1}, {1, 1, 0} → {0, 1},
 {1, 0, 1} → {1, -1}, {1, 0, 0} → {0, -1}, {0, 1, 1} → {0, -1},
 {0, 1, 0} → {0, 1}, {0, 0, 1} → {1, 1}, {0, 0, 0} → {1, -1}}
```

where the left-hand side in each case gives the value of the active cell and its left and right neighbors, while the right-hand side consists of a pair containing the new value of the active cell and the displacement of its position. (In analogy with cellular automata, this rule can be labelled *{35, 57}* where the first number refers to colors, and the second displacements.) With a rule given in this form, each step in the evolution of the mobile automaton corresponds to the function

```
MAStep[rule_, {list_List, n_Integer}] /; 1 < n < Length[list] :=
    Apply[{ReplacePart[list, #1, n], n + #2} &,
        Replace[Take[list, {n - 1, n + 1}], rule]]
```

The complete evolution for many steps can then be obtained with

```
MAEvolveList[rule_, init_List, t_Integer] :=
  NestList[MAStep[rule, #] &, init, t]
```

(The program will run more efficiently if *Dispatch* is applied to the rule before giving it as input.)

For the mobile automaton on page 73, the rule can be given as

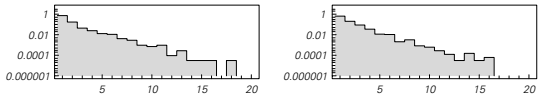
```
{{1, 1, 1} → {{0, 0, 0}, -1}, {1, 1, 0} → {{1, 0, 1}, -1},
 {1, 0, 1} → {{1, 1, 1}, 1}, {1, 0, 0} → {{1, 0, 0}, 1},
 {0, 1, 1} → {{0, 0, 0}, 1}, {0, 1, 0} → {{0, 1, 1}, -1},
 {0, 0, 1} → {{1, 0, 1}, 1}, {0, 0, 0} → {{1, 1, 1}, 1}}
```

and *MAStep* must be rewritten as

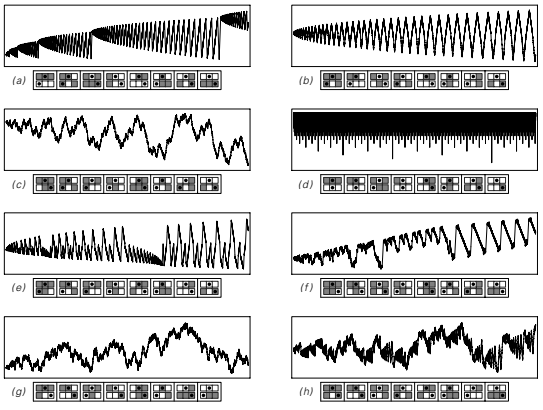
```
MAStep[rule_, {list_List, n_Integer}] /; 1 < n < Length[list] :=
  Apply[{Join[Take[list, {1, n - 2}], #1, Take[list, {n + 2, -1}]],
    n + #2} &, Replace[Take[list, {n - 1, n + 1}], rule]]
```

■ **Compressed evolution.** An alternative compression scheme for mobile automata is discussed on page 488.

■ **Page 72 · Distribution of behavior.** The pictures below show the distributions of transient and of period lengths for the 65,318 mobile automata of the type described here that yield ultimately repetitive behavior. Rule (f) has a period equal to the maximum of 16.



■ **Page 75 · Active cell motion.** The pictures below show the positions of the active cell for 20,000 steps of evolution in various mobile automata. (a), (b) and (c) correspond respectively to the rules on pages 73, 74 and 75. (c) has an outer envelope whose edges grow at rates $\{-1.5, 0.3\}\sqrt{t}$. (d) yields logarithmic growth as shown on page 496 (like Turing machine (f) on page 79). In most cases where the behavior is ultimately repetitive, transients and periods seem to follow the same approximate exponential distribution as in the note above. (g) however suddenly yields repetitive behavior with period 4032 after 405,941 steps. (h) does not appear to evolve to strict repetition or nesting, but does show progressively longer patches with fairly orderly behavior. (c) shows no obvious deviation from randomness in at least the first billion steps (after which the pattern it produces is 57,014 cells wide).



■ **Implementation of generalized mobile automata.** The state of a generalized mobile automaton at a particular step can be

specified by *{list, nlist}*, where *list* gives the values of the cells, and *nlist* is a list of the positions of active cells. The rule can be given by specifying a list of cases such as *{0, 0, 0}→{1, {1, -1}}*, where in each case the second sublist specifies the new relative positions of active cells. With this setup successive steps in the evolution of the system can be obtained from

```
GMAStep[rules_, {list_, nlist_}] := Module[{a, na}, {a, na} =
  Transpose[Map[Replace[Take[list, {# - 1, # + 1}], rules] &,
   nlist]]; {Fold[ReplacePart[#1, Last[#2], First[#2]] &,
   list, Transpose[{nlist, a}]], Union[Flatten[nlist + na]]}]
```

Turing Machines

■ **Implementation.** The state of a Turing machine at a particular step can be represented by the triple *{s, list, n}*, where *s* gives the state of the head, *list* gives the values of the cells, and *n* specifies the position of the head (the cell under the head thus has value *list[[n]]*). Then, for example, the rule for the Turing machine shown on page 78 can be given as

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

where the left-hand side in each case gives the state of the head and the value of the cell under the head, and the right-hand side consists of a triple giving the new state of the head, the new value of the cell under the head and the displacement of the head.

With a rule given in this form, a single step in the evolution of the Turing machine can be implemented with the function

```
TMStep[rule_List, {s_, a_List, n_}] /; 1 ≤ n ≤ Length[a] :=
  Apply[{#1, ReplacePart[a, #2, n], n + #3} &,
   Replace[{s, a[[n]]}, rule]]
```

The evolution for many steps can then be obtained using

```
TMEvolveList[rule_, init_List, t_Integer] :=
  NestList[TMStep[rule, #] &, init, t]
```

An alternative approach is to represent the complete state of the Turing machine by *MapAt[{s, #} &, list, n]*, and then to use

```
TMStep[rule_, c_] := Replace[c,
  {a___, x_, h_List, y_, b___} :> Apply[{{a, x, #2, {#1, y}, b},
    {a, {#1, x}, #2, y, b}}[[#3]] &, h /. rule]]
```

The result of *t* steps of evolution from a blank tape can also be obtained from (see also page 1143)

```
s = 1; a[_] = 0; n = 0;
Do[{s, a[n], d} = {s, a[n]} /. rule; n += d, {t}]
```

■ **Number of rules.** With *k* possible colors for each cell and *s* possible states, there are a total of $(2sk)^{sk}$ possible Turing machine rules. Often many of these rules are immediately equivalent, or can show only very simple behavior (see page 1120).

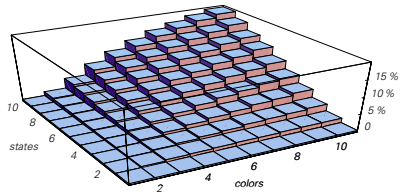
■ **Numbering scheme.** One can number Turing machines and get their rules using

```
Flatten[MapIndexed[{{1, -1}#2 + {0, k} → {1, 1, 2}
    Mod[Quotient[#1, {2 k, 2, 1}], {s, k, 2}] + {1, 0, -1} &,
  Partition[IntegerDigits[n, 2 s k, s k], k], {2}]]
```

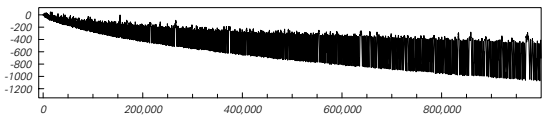
The examples on page 79 have numbers 3024, 982, 925, 1971, 2506 and 1953.

■ **Page 79 · Counter machine.** Turing machine (f) operates like a base 2 counter: at steps where its head is at the leftmost position, the colors of the cells correspond to the reverse of the base 2 digit sequences of successive numbers. All possible arrangements of colors are thus eventually produced. The overall pattern attains width *j* after $2^j - j$ steps.

■ **Page 80 · Distribution of behavior.** With 2 possible states and 2 possible colors for each cell, starting from a blank tape, the maximum repetition period obtained is 9 steps, and 12 out of the 4096 possible rules (or about 0.29%) yield non-repetitive behavior. With 3 states and 2 colors, the maximum period is 24, and about 0.37% of rules yield non-repetitive behavior, always nested. (Usually I have not found more complicated behavior in such rules even with initial conditions in which there are both black and white cells, though see page 761.) With 2 states and 3 colors, the maximum repetition period is again 24, about 0.65% of rules yield non-repetitive behavior, and the 14 rules discussed on page 709 yield more complex behavior. With more colors or more states, the percentage of rules that yield non-repetitive behavior steadily increases, as shown below, roughly like $0.28(s-1)(k-1)$. (Compare page 1120.)

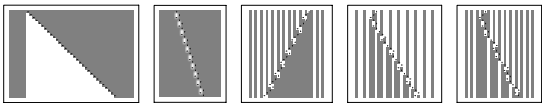


■ **Page 81 · Head motion.** The picture below shows the motion of the head for the first million steps. After about 20,000 steps, the width of the pattern produced grows at a rate close to $\sqrt{t}$.



■ **Localized structures.** Even when the overall behavior of a Turing machine is complicated, it is possible for simple localized structures to exist, much as in cellular automata

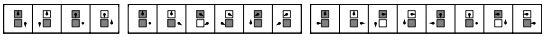
such as rule 110. What can happen is that with certain specific repetitive backgrounds, the head can move in a simple repetitive way, as shown in the pictures below for the Turing machine from page 81.



■ **History.** Turing machines were invented by Alan Turing in 1936 to serve as idealized models for the basic processes of mathematical calculation (see page 1128). As discussed on page 1110, Turing's main interest was in showing what his machines could in principle be made to do, not in finding out what simple examples of them actually did. Indeed, so far as I know, even though he had access to the necessary technology, Turing never explicitly simulated any Turing machine on a computer.

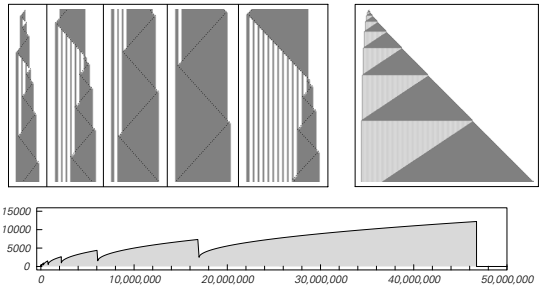
Since Turing's time, Turing machines have been extensively used as abstract models in theoretical computer science. But in almost no cases has the explicit behavior of simple Turing machines been considered. In the early 1960s, however, Marvin Minsky and others did work on finding the simplest Turing machines that could exhibit certain properties. Most of their effort was devoted to finding ingenious constructions for creating appropriate machines (see page 1119). But around 1961 they did systematically study all 4096 2-state 2-color machines, and simulated the behavior of some simple Turing machines on a computer. They found repetitive and nested behavior, but did not investigate enough examples to discover the more complex behavior shown in the main text.

As an offshoot of abstract studies of Turing machines, Tibor Radó in 1962 formulated what he called the Busy Beaver Problem: to find a Turing machine with a specified number of states that "keeps busy" for as many steps as possible before finally reaching a particular "halt state" (numbered 0 below). (A variant of the problem asks for the maximum number of black cells that are left when the machine halts.) By 1966 the results for 2, 3 and 4 states had been found: the maximum numbers of steps are 6, 21 and 107, respectively, with 4, 5 and 13 final black cells. Rules achieving these bounds are:



The result for 5 states is still unknown, but a machine taking 47,176,870 steps and leaving 4098 black cells was found by Heiner Marxen and Jürgen Buntrock in 1990. Its rule is:

The pictures below show (a) the first 500 steps of evolution, (b) the first million steps in compressed form and (c) the number of black cells obtained at each step. Perhaps not surprisingly for a system optimized to run as long as possible, the machine operates in a rather systematic and regular way. With 6 states, a machine is known that takes about 3.002×10^{1730} steps to halt, and leaves about 1.29×10^{865} black cells. (See also page 1144.)



Substitution Systems

■ **Implementation.** The rule for a neighbor-independent substitution system such as the first one on page 82 can conveniently be given as *{1→{1, 0}, 0→{0, 1}}*. And with this representation, the evolution for *t* steps is given by

```
SSEvolveList[rule_, init_List, t_Integer] :=
  NestList[Flatten[# /. rule] &, init, t]
```

where in the first example on page 82, the initial condition is *{1}*.

An alternative approach is to use strings, representing the rule by *{"B"→"BA", "A"→"AB"}* and the initial condition by *"B"*. In this case, the evolution can be obtained using

```
SSEvolveList[rule_, init_String, t_Integer] :=
  NestList[StringReplace[#, rule] &, init, t]
```

For a neighbor-dependent substitution system such as the first one on page 85 the rule can be given as

```
{{1, 1}→{0, 1}, {1, 0}→{1, 0}, {0, 1}→{0}, {0, 0}→{0, 1}}
```

And with this representation, the evolution for *t* steps is given by

```
SS2EvolveList[rule_, init_List, t_Integer] :=
  NestList[Flatten[Partition[#, 2, 1] /. rule] &, init, t]
```

where the initial condition for the first example on page 85 is *{0, 1, 1, 0}*.

■ **Page 83 · Properties.** The examples shown here all appear in quite a number of different contexts in this book. Note that each of them in effect yields a single sequence that gets progressively longer at each step; other rules make the colors of elements alternate on successive steps.

(a) *(Successive digits sequence)* The sequence produced is repetitive, with the element at position *n* being black for *n*

odd and white for n even. There are a total of 2^t elements after t steps. The complete pattern formed by looking at all the steps together has the same structure as the arrangement of base 2 digits in successive numbers shown on page 117.

(b) *(Thue-Morse sequence)* The color *s[n]* of the element at position n is given by *1 - Mod[DigitCount[n - 1, 2, 1], 2]*. These colors satisfy *s[n_] := If[EvenQ[n], 1 - s[n/2], s[(n + 1)/2]]* with *s[1] = 1*. There are a total of 2^t elements in the sequence after t steps. The sequence on step t can be obtained from *Nest[Join[#, 1 - #] &, {1}, t - 1]*. The number of black and white elements at each step is always the same. All four possible pairs of successive elements occur, though not with equal frequency. Runs of three identical elements never occur, and in general no block of elements can ever occur more than twice. The first 2^m elements in the sequence can be obtained from (see page 1081)

```
(CoefficientList[Product[1 - z^2^s, {s, 0, m - 1}], z] + 1)/2
```

The first n elements can also be obtained from (see page 1092)

```
Mod[CoefficientList[Series[(1 + Sqrt[(1 - 3 x)/(1 + x)])/
    (2 (1 + x)), {x, 0, n - 1}], x], 2]
```

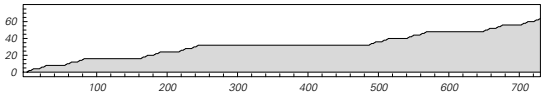
The sequence occurs many times in this book; it can for example be derived from a column of values in the rule 150 cellular automaton pattern discussed on page 885.

(c) *(Fibonacci-related sequence)* The sequence at step t can be obtained from *a[t_] := Join[a[t - 1], a[t - 2]]; a[1] = {0}; a[2] = {0, 1}*. This sequence has length *Fibonacci[t + 1]* (or approximately 1.618^{t+1}) (see note below). The color of the element at position n is given by *2 - (Floor[(n + 1) GoldenRatio] - Floor[n GoldenRatio])* (see page 904), while the position of the k^{th} white element is given by the so-called Beatty sequence *Floor[k GoldenRatio]*. The ratio of the number of white elements to black at step t is *Fibonacci[t - 1]/Fibonacci[t - 2]*, which approaches *GoldenRatio* for large t. For all $m \le$ *Fibonacci[t - 1]*, the number of distinct blocks of m successive elements that actually appear out of the 2^m possibilities is $m + 1$ (making it a so-called Sturmian sequence as discussed on page 1084).

(d) *(Cantor set)* The color of the element at position n is given by *If[FreeQ[IntegerDigits[n - 1, 3], 1], 1, 0]*, which turns out to be equivalent to

```
If[OddQ[n], Sign[Mod[Binomial[n - 1, (n - 1)/2], 3]], 0, 1]
```

There are 3^t elements after t steps, of which 2^t are black. The picture below shows the number of black cells that occur before position n. The resulting curve has a nested form, with envelope *n^Log[3, 2]*.



■ **Growth rates.** The total number of elements of each color that occur at each step in a neighbor-independent substitution system can be found by forming the matrix m where *m[[i, j]]* gives the number of elements of color *j + 1* that appear in the block that replaces an element of color *i + 1*. For case (c) above, *m = {{1, 1}, {1, 0}}*. A list that gives the number of elements of each color at step t can then be found from *init . MatrixPower[m, t]*, where *init* gives the initial number of elements of each color—*{1, 0}* for case (c) above. For large t, the total number of elements typically grows like λ^t, where λ is the largest eigenvalue of m; the relative numbers of elements of each color are given by the corresponding eigenvector. For case (c), λ is *GoldenRatio*, or $(1 + \sqrt{5})/2$. There are exceptional cases where $\lambda == 1$, so that the growth is not exponential. For the rule *{0 → {0, 1}, 1 → {1}}*, *m = {{1, 1}, {0, 1}}*, and the number of elements at step t starting with *{0}* is just t. For *{0 → {0, 1}, 1 → {1, 2}, 2 → {2}}*, *m = {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}}*, and the number of elements starting with *{0}* is $(t^2 - t + 2)/2$. For neighbor-independent rules, the growth for large t must follow an exponential or an integer power less than the number of possible colors. For neighbor-dependent rules, any form of growth can in principle be obtained.

■ **Fibonacci numbers.** The Fibonacci numbers *Fibonacci[n]* (*f[n]* for short) can be generated by the recurrence relation

```
f[n_] := f[n] = f[n - 1] + f[n - 2]
f[1] = f[2] = 1
```

The first few Fibonacci numbers are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377. For large n the ratio *f[n]/f[n - 1]* approaches *GoldenRatio* or $(1 + \sqrt{5})/2 \simeq 1.618$.

Fibonacci[n] can be obtained in many ways:

- $(\text{GoldenRatio}^n - (-\text{GoldenRatio})^{-n})/\sqrt{5}$
- *Round[GoldenRatio^n/Sqrt[5]]*
- 2^{1-n} *Coefficient[(1 + Sqrt[5])^n, Sqrt[5]]*
- *MatrixPower[{{1, 1}, {1, 0}}, n - 1][[1, 1]]*
- *Numerator[NestList[1/(1 + #) &, 1, n]]*
- *Coefficient[Series[1/(1 - t - t^2), {t, 0, n}], t^(n-1)]*
- *Sum[Binomial[n - i - 1, i], {i, 0, (n - 1)/2}]*
- 2^{n-2} *- Count[IntegerDigits[Range[0, 2^(n-2)], 2], {___, 1, 1, ___}]*

A fast method for evaluating *Fibonacci[n]* is

```
First[Fold[f, {1, 0, -1}, Rest[IntegerDigits[n, 2]]]]
f[{a_, b_, s_}, 0] = {a (a + 2 b), s + a (2 a - b), 1}
f[{a_, b_, s_}, 1] = {-s + (a + b) (a + 2 b), a (a + 2 b), -1}
```

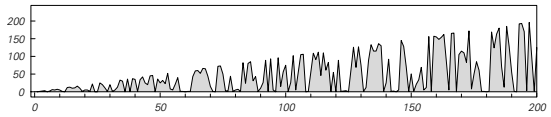
Fibonacci numbers appear to have first arisen in perhaps 200 BC in work by Pingala on enumerating possible patterns of

poetry formed from syllables of two lengths. They were independently discussed by Leonardo Fibonacci in 1202 as solutions to a mathematical puzzle concerning rabbit breeding, and by Johannes Kepler in 1611 in connection with approximations to the pentagon. Their recurrence relation appears to have been understood from the early 1600s, but it has only been in the past very few decades that they have in general become widely discussed.

For $m > 1$, the value of n for which m == *Fibonacci[n]* is *Round[Log[GoldenRatio, $\sqrt{5}$ m]]*.

The sequence *Mod[Fibonacci[n], k]* is always purely repetitive; the maximum period is $6k$, achieved when $k = 10\,5^m$ (compare page 975).

Mod[Fibonacci[n], n] has the fairly complicated form shown below. It appears to be zero only when n is of the form 5^m or $12q$, where q is not prime ($q > 5$).



The number *GoldenRatio* appears to have been used in art and architecture since antiquity. *1/GoldenRatio* is the default *AspectRatio* for *Mathematica* graphics. In addition:

- *GoldenRatio* is the solution to x == $1 + 1/x$ or x^2 == $x + 1$
- The right-hand rectangle in ▭ is similar to the whole rectangle when the aspect ratio is *GoldenRatio*
- *Cos[π/5] == Cos[36°] == GoldenRatio/2*
- The ratio of the length of the diagonal to the length of a side in a regular pentagon is *GoldenRatio*
- The corners of an icosahedron are at coordinates
  Flatten[Array[NestList[RotateRight, {0, (−1)^#1 GoldenRatio, (−1)^#2}, 3] &, {2, 2}], 2]
- *1 + FixedPoint[N[1/(1 + #), k] &, 1]* approximates *GoldenRatio* to k digits, as does *FixedPoint[N[Sqrt[1 + #], k] &, 1]*
- A successive angle difference of *GoldenRatio* radians yields points maximally separated around a circle (see page 1006).

■ **Lucas numbers.** Lucas numbers *Lucas[n]* satisfy the same recurrence relation *f[n_] := f[n − 1] + f[n − 2]* as Fibonacci numbers, but with the initial conditions *f[1] = 1; f[2] = 3*. Among the relations satisfied by Lucas numbers are:

- *Lucas[n_] := Fibonacci[n − 1] + Fibonacci[n + 1]*
- *GoldenRatio^n == (Lucas[n] + Fibonacci[n] $\sqrt{5}$)/2*

■ **Generalized Fibonacci sequences.** Any linear recurrence relation yields sequences with many properties in common with the Fibonacci numbers—though with *GoldenRatio* replaced by other algebraic numbers. The Perrin sequence *f[n_] := f[n − 2] + f[n − 3]; f[0] = 3; f[1] = 0; f[2] = 2* has the peculiar property that *Mod[f[n], n] == 0* mostly but not always only for n prime. (For more on recurrence relations see page 128.)

■ **Connections with digit sequences.** In a sequence generated by a neighbor-independent substitution system the color of the element at position n turns out always to be related to the digit sequence of the number n in an appropriate base. The basic reason for this is that as shown on page 84 the evolution of the substitution system always yields a tree, and the successive digits in n determine which branch is taken at each level in order to reach the element at position n. In cases (a) and (b) on pages 83 and 84, the tree has two branches at every node, and so the base 2 digits of n determine the successive left and right branches that must be taken. Given that a branch with a certain color has been reached, the color of the branch to be taken next is then determined purely by the next digit in the digit sequence of n. For case (b) on pages 83 and 84, the rule that gives the color of the next branch in terms of the color of the current branch and the next digit is *{{0, 0} → 0, {0, 1} → 1, {1, 0} → 1, {1, 1} → 0}*. In terms of this rule, the color of the element at position n is given by

Fold[Replace[{#1, #2}, rule] &, 1, IntegerDigits[n − 1, 2]]

The rule used here can be thought of as a finite automaton with two states. In general, the behavior of any neighbor-independent substitution system where each element is subdivided into exactly k elements can be reproduced by a finite automaton with k states operating on digit sequences in base k. The nested structure of the patterns produced is thus a direct consequence of the nesting seen in the patterns of these digit sequences, as shown on page 117.

Note that if the rule for the finite automaton is represented for example as *{{1, 2}, {2, 1}}* where each sublist corresponds to a particular state, and the elements of the sublist give the successor states with inputs *Range[0, k − 1]*, then the n^{th} element in the output sequence can be obtained from

Fold[rule[[#1, #2]] &, 1, IntegerDigits[n − 1, k] + 1] − 1

while the first k^m elements can be obtained from

Nest[Flatten[rule[[#]]] &, 1, m] − 1

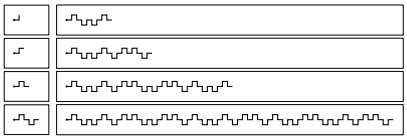
To treat examples such as case (c) where elements can subdivide into blocks of several different lengths one must generalize the notion of digit sequences. In base k a number is constructed from a digit sequence *a[r], …, a[1], a[0]* (with $0 \le a[i] < k$) according to *Sum[a[i] k^i, {i, 0, r}]*. But given a sequence of digits that are each 0 or 1, it is also possible for example to construct numbers according to

Sum[a[i] Fibonacci[i + 2], {i, 0, r}]. (As discussed on page 1070, this representation is unique so long as one does not allow any pairs of adjacent 1's in the digit sequence.) It then turns out that if one expresses the position *n* as a generalized digit sequence of this kind, then the color of the corresponding element in substitution system (c) is just the last digit in this sequence.

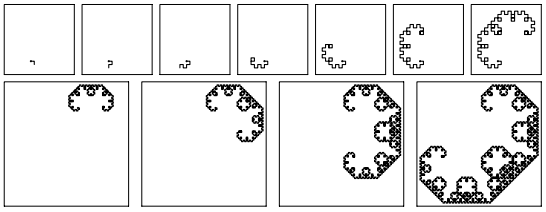
■ **Connections with square roots.** Substitution systems such as (c) above are related to projections of lines with quadratic irrational slopes, as discussed on page 904.

■ **Spectra of substitution systems.** See page 1080.

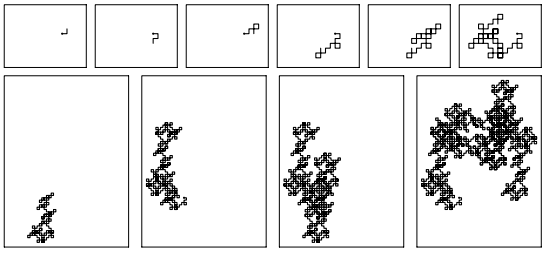
■ **Representation by paths.** An alternative to representing substitution systems by 1D sequences of black and white squares is to use 2D paths consisting of sequences of left and right turns. The paths obtained at successive steps for rule (b) above are shown below.



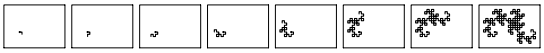
The pictures below show paths obtained with the rule *{1 → {1}, 0 → {0, 0, 1}}*, starting from *{0}*. Note the similarity to the 2D system shown on page 190.



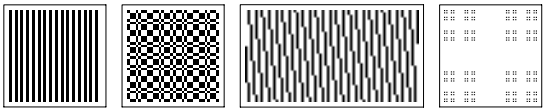
When the paths do not cross themselves, nested structure is evident. But in a case like the rule *{1 → {0, 0, 1}, 0 → {1, 0}}* starting with *{1}*, the presence of many crossings tends to hide such regularity, as in the pictures below.



■ **Paperfolding sequences.** The sequence of up and down creases in a strip of paper that is successively folded in half is given by a substitution system; after *t* steps the sequence turns out to be *NestList[Join[#, {0}, Reverse[1 - #]] &, {0}, t]*. The corresponding path (effectively obtained by making each crease a right angle) is shown below. (See page 189.)



■ **2D representations.** Individual sequences from 1D substitution systems can be displayed in 2D by breaking them into a succession of rows. The pictures below show results for the substitution systems on page 83. In case (b), with rows chosen to be 2^j elements in length, the leftmost column will always be identical to the beginning of the sequence, and in addition every interior element will be black exactly when the cell at the top of its column has the same color as the one at the beginning of its row. In case (c), stripes appear at angles related to *GoldenRatio*.



■ **Page 84 · Other examples.**

(a) *(Period-doubling sequence)* After *t* steps, there are a total of 2^t elements, and the sequence is given by *Nest[MapAt[1 - # &, Join[#, #], -1] &, {0}, t]*. It contains a total of *Round[2^t/3]* black elements, and if the last element is dropped, it forms a palindrome. The n^{th} element is given by *Mod[IntegerExponent[n, 2], 2]*. As discussed on page 885, the sequence appears in a vertical column of cellular automaton rule 150. The Thue-Morse sequence discussed on page 890 can be obtained from it by applying

```
1 - Mod[Flatten[Partition[FoldList[Plus, 0, list], 1, 2]], 2]
```

(b) The n^{th} element is simply *Mod[n, 2]*.

(c) Same as (a), after the replacement *1 → {1, 1}* in each sequence. Note that the spectra of (a) and (c) are nevertheless different, as discussed on page 1080.

(d) The length of the sequence at step *t* satisfies *a[t] == 2 a[t - 1] + a[t - 2]*, so that *a[t] = Round[(1 + √2)^(t-1)/2]* for *t > 1*. The number of white elements at step *t* is then *Round[a[t]/√2]*. Much like example (c) on page 83 there are *m + 1* distinct blocks of length *m*, and with *f = Floor[(1 - 1/√2)(# + 1/√2)] &* the n^{th} element of the sequence is given by *f[n + 1] - f[n]* (see page 903).

(e) For large t the number of elements increases like λ^t with $\lambda = (\sqrt{13} + 1)/2$; there are always λ times as many white elements as black ones.

(f) The number of elements at step t is $Round[(1 + \sqrt{2})^t/2]$, and the n^{th} element is given by $Floor[\sqrt{2}\,(n+1)] - Floor[\sqrt{2}\,n]$ (see page 903).

(g) The number of elements is the same as in (f).

(h) The number of black elements is 2^{t-1}; the total number of elements is $2^{t-2}\,(t+1)$.

(i) and (j) The total number of elements is 3^{t-1}.

■ **History.** In their various representations, 1D substitution systems have been invented independently many times for many different purposes. (For the history of fractals and 2D substitution systems see page 934.) Viewed as generators of sequences with certain combinatorial properties, substitution systems such as example (b) on page 83 appeared in the work of Axel Thue in 1906. (Thue's stated purpose in this work was to develop the science of logic by finding difficult problems with possible connections to number theory.) The sequence of example (b) was rediscovered by Marston Morse in 1917 in connection with his development of symbolic dynamics—and in finding what could happen in discrete approximations to continuous systems. Studies of general neighbor-independent substitution systems (sometimes under such names as sequence homomorphisms, iterated morphisms and uniform tag systems) have continued in this context to this day. In addition, particularly since the 1980s, they have been studied in the context of formal language theory and the so-called combinatorics of words. (Period-doubling phenomena also led to contact with physics starting in the late 1970s.)

Independent of work in symbolic dynamics, substitution systems viewed as generators of sequences were reinvented in 1968 by Aristid Lindenmayer under the name of L systems for the purpose of constructing models of branching plants (see page 1005). So-called 0L systems correspond to my neighbor-independent substitution systems; 1L systems correspond to the neighbor-dependent substitution systems on page 85. Work on L systems has proceeded along two quite different lines: modelling specific plant systems, and investigating general computational capabilities. In the mid-1980s, particularly through the work of Alvy Ray Smith, L systems became widely used for realistic renderings of plants in computer graphics.

The idea of constructing abstract trees such as family trees according to definite rules presumably goes back to antiquity. The tree representation of rule (c) from page 83 was for example probably drawn by Leonardo Fibonacci in 1202.

The first six levels of the specific pattern in example (a) on page 83 correspond exactly to the segregation diagram for the I Ching that arose in China as early as 2000 BC. Black regions represent yin and white ones yang. The elements on level six correspond to the 64 hexagrams of the I Ching. At what time the segregation diagram was first drawn is not clear, but it was almost certainly before 1000 AD, and in the 1600s it appears to have influenced Gottfried Leibniz in his development of base 2 numbers.

Viewed in terms of digit sequences, example (d) from page 83 was discussed by Georg Cantor in 1883 in connection with his investigations of the idea of continuity. General relations between digit sequences and sequences produced by neighbor-independent substitution systems were found in the 1960s. Connections of sequences such as (c) to algebraic numbers (see page 903) arose in precursors to studies of wavelets.

Paths representing sequences from 1D substitution systems can be generated by 2D geometrical substitution systems, as on page 189. The "C" curve shown on the facing page and on page 190 was for example described by Paul Lévy in 1937, and was rediscovered as the output of a simple computer program by William Gosper in the 1960s. Paperfolding or so-called dragon curves (as shown above) were discussed by John Heighway in the mid-1960s, and were analyzed by Chandler Davis, Donald Knuth and others. These curves have the property that they eventually fill space. Space-filling curves based on slightly more complicated substitution systems were already discussed by Giuseppe Peano in 1890 and by David Hilbert in 1891 in connection with questions about the foundations of calculus.

Sequences from substitution systems have no doubt appeared over the years as incidental features of great many pieces of mathematical work. As early as 1851, for example, Eugène Prouhet showed that if sequences of integers were partitioned according to sequence (b) on page 83, then sums of powers of these integers would be equal: thus $Apply[Plus, Flatten[Position[s, i]]^k]$ is equal for $i = 0$ and $i = 1$ if s is a sequence of the form (b) on page 83 with length 2^m, $m > k$. The optimal solution to the Towers of Hanoi puzzle invented in 1883 also turns out to be an example of a substitution system sequence.

Sequential Substitution Systems

■ **Implementation.** Sequential substitution systems can be implemented quite directly by using *Mathematica*'s standard

mechanism for applying transformation rules to symbolic expressions. Having made the definition

```
Attributes[s] = Flat
```

the state of a sequential substitution system at a particular step can be represented by a symbolic expression such as `s[1, 0, 1, 0]`. The rule on page 82 can then be given simply as

```
s[1, 0] → s[0, 1, 0]
```

while the rule on page 85 becomes

```
{s[0, 1, 0] → s[0, 0, 1], s[0] → s[0, 1, 0]}
```

The *Flat* attribute of `s` makes these rules apply not only for example to the whole sequence `s[1, 0, 1, 0]` but also to any subsequence such as `s[1, 0]`. (With `s` being *Flat*, `s[s[1, 0], 1, s[0]]` is equivalent to `s[1, 0, 1, 0]` and so on. A *Flat* function has the mathematical property of being associative.) And with this setup, `t` steps of evolution can be found with

```
SSSEvolveList[rule_, init_s, t_Integer] :=
  NestList[# /. rule &, init, t]
```

Note that as an alternative to having `s` be *Flat*, one can explicitly set up rules based on patterns such as `s[x___, 1, 0, y___] → s[x, 0, 1, 0, y]`. And by using rules such as `s[x___, 1, 0, y___] :> {s[x, 0, 1, 0, y], Length[s[x]]}` one can keep track of the positions at which substitutions are made. (*StringReplace* replaces all occurrences of a given substring, not just the first one, so cannot be used directly as an alternative to having a flat function.)

■ **Capabilities.** Even with the single rule `{s[1, 0] → s[0, 1]}`, a sequential substitution system can sort its initial conditions so that all 0's occur before all 1's. (See also page 1113.)

■ **Order of replacements.** For many sequential substitution systems the evolution effectively stops because a string is produced to which none of the replacements given apply. In most sequential substitution systems there is more than one possible replacement that can in principle apply at a particular step, so the order in which the replacements are tried matters. (Multiway systems discussed on page 497 are what result if all possible replacements are performed at each step.) There are however special sequential substitution systems (those with the so-called confluence property discussed on page 1036) in which in a certain sense the order of replacements does not matter.

■ **History.** Sequential substitution systems are closely related to the multiway systems discussed on page 938, and are often considered examples of production systems or string rewriting systems. In the form I discuss here, they seem to have arisen first under the name "normal algorithms" in the work of Andrei Markov in the late 1940s on computability and the idealization of mathematical processes. Starting in the 1960s text editors like TECO and ed used sequential substitution system rules, as have string-processing languages such as SNOBOL and perl. *Mathematica* uses an analog of sequential substitution system rules to transform general symbolic expressions. The fact that new rules can be added to a sequential substitution system incrementally without changing its basic structure has made such systems popular in studies of adaptive programming.

Tag Systems

■ **Implementation.** With the rules for case (a) on page 94 given for example by

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

the evolution of a tag system can be obtained from

```
TSEvolveList[{n_, rule_}, init_, t_] := NestList[If[Length[#] <
    n, {}, Join[Drop[#, n], Take[#, n] /. rule]] &, init, t]
```

An alternative implementation is based on applying to the list at each step rules such as

```
{{0, 0, s___} → {s, 1, 1}, {1, 0, s___} → {s},
  {0, 1, s___} → {s, 1, 0}, {1, 1, s___} → {s, 0, 0, 0}}
```

There are a total of $((k^{r+1} - 1)/(k - 1))^{k^n}$ possible rules if blocks up to length r can be added at each step and k colors are allowed. For $r = 3$, $k = 2$ and $n = 2$ this is 50,625.

■ **Page 94 · Randomness.** To get some idea of the randomness of the behavior, one can look at the sequence of first elements produced on successive steps. In case (a), the fraction of black elements fluctuates around 1/2; in (b) it approaches 3/4; in (d) it fluctuates around near 0.3548, while in (e) and (f) it does not appear to stabilize.

■ **History.** The tag systems that I consider are generalizations of those first discussed by Emil Post in 1920 as simple idealizations of certain syntactic reduction rules in Alfred Whitehead and Bertrand Russell's *Principia Mathematica* (see page 1149). Post's tag systems differ from mine in that his allow the choice of block that is added at each step to depend only on the very first element in the sequence at that step (see however page 670). (The lag systems studied in 1963 by Hao Wang allow dependence on more than just the first element, but remove only the first element.) It turns out that in order to get complex behavior in such systems, one needs either to allow more than two possible colors for each element, or to remove more than two elements from the beginning of the sequence at each step. Around 1921, Post apparently studied all tag systems of his type that involve removal and addition of no more than two elements at each step, and he concluded that none of them produced complicated behavior. But then he looked at rules that

remove three elements at each step, and he discovered the rule *{3, {{0, _, _}→{0, 0}, {1, _, _}→{1, 1, 0, 1}}}*. As he noted, the behavior of this rule varies considerably with the initial conditions used. But at least for all the initial conditions up to length 28, the rule eventually just leads to behavior that repeats with a period of 1, 2, 6, 10, 28 or 40. With more than two colors, one finds that rules of Post's type which remove just two elements at each step can yield complex behavior, even starting from an initial condition such as *{0, 0}*. An example is *{2, {{0, _}→{2, 1}, {1, _}→{0}, {2, _}→{0, 2, 1, 2}}}*. (See also pages 1113 and 1141.)

Cyclic Tag Systems

■ **Implementation.** With the rules for the cyclic tag system on page 95 given as *{{1, 1}, {1, 0}}*, the evolution can be obtained from

```
CTEvolveList[rules_, init_, t_] :=
  Map[Last, NestList[CTStep, {rules, init}, t]]
CTStep[{{r_, s___}, {0, a___}}] := {{s, r}, {a}}
CTStep[{{r_, s___}, {1, a___}}] := {{s, r}, Join[{a}, r]}
CTStep[{u_, {}}] := {u, {}}
```

The leading elements on many more than t successive steps can be obtained directly from

```
CTList[rules_, init_, t_] :=
  Flatten[Map[Last, NestList[CTListStep, {rules, init}, t]]]
CTListStep[{rules_, list_}] :=
  {RotateLeft[rules, Length[list]], Flatten[rules[[
      Mod[Flatten[Position[list, 1]], Length[rules], 1]]]]}
```

■ **Page 95 · Generalizations.** The implementation above immediately allows cyclic tag systems which cycle through a list of more than two blocks. (With just one block the behavior is always repetitive.) Cyclic tag systems which allow any value for each element can be obtained by adding the rule

```
CTStep[{{r_, s___}, {n_, a___}}] :=
  {{s, r}, Flatten[{a, Table[r, {n}]}]}
```

The leading elements in this case can be obtained using

```
CTListStep[{rules_, list_}] :=
 {RotateLeft[rules, Length[list]], With[{n = Length[rules]},
    Flatten[Apply[Table[#1, {#2}] &, Map[Transpose[
        {rules, #}] &, Partition[list, n, n, 1, 0]], {2}]]]}
```

■ **Mechanical implementation.** Cyclic tag systems admit a particularly straightforward mechanical implementation. Black and white balls are kept in a trough as in the picture below. At each step the leftmost ball in the trough is released, and if this ball is black (as determined, for example, by size) a mechanism causes a new block of balls to be added at the right-hand end of the trough. This mechanism can work in several ways; typically it will involve a rotary element that determines which case of the rule to use at each step. Rule (e) from the main text allows a particularly simple supply of new balls. Note that the system will inevitably fail if the trough overflows with balls.

■ **Page 96 · Properties.** Assuming that black and white elements occur in an uncorrelated way, then the sequences in a cyclic tag system with n blocks should grow by an average of *Count[Flatten[rules], 1]/n - 1* elements at each step. With $n = 2$ blocks, this means that growth can occur only if the total number of black elements in both blocks is more than 3. Rules such as *{{1, 0}, {0, 1}}* and *{{1, 1}, {0}}* therefore yield repetitive behavior with sequences of limited length.

Note that if all blocks in a cyclic tag system with n blocks have lengths divisible by n, then one can tell in advance on which steps blocks will be added, and the overall behavior obtained must correspond to a neighbor-independent substitution system. The rules for the relevant substitution system may however depend on the initial conditions for the cyclic tag system.

Flatten[{1, 0, CTList[{{1, 0, 0, 1}, {0, 1, 1, 0}}, {0, 1}, t]}]

gives for example the Thue-Morse substitution system *{1→{1, 0}, 0→{0, 1}}*.

In example (a), the elements are correlated, so that slower growth occurs than in the estimate above. In example (c), the elements are again correlated: the growth is by an average of $(\sqrt{5}-1)/2 \approx 0.618$ elements at each step, and the first elements on alternate steps form the same nested sequence as obtained from the substitution system *{1→{1, 0}, 0→{1}}*. In example (d), the frequency of 1's among the first elements of sequence is approximately 3/4; *{0, 0}* never occurs, and the frequency of *{1, 1}* is approximately 1/2. In example (e), the frequency of 1's is again about 3/4, but now *{0, 0}* occurs with frequency 0.05, *{1, 1}* occurs with frequency 0.55, while *{0, 0, 0}* and *{0, 1, 0}* cannot occur.

■ **History.** Cyclic tag systems were studied by Matthew Cook in 1994 in connection with working on the rule 110 cellular automaton for this book. The sequence *{1, 2, 2, 1, 1, 2, …}* defined by the property *list == Map[Length, Split[list]]* was suggested as a mathematical puzzle by William Kolakoski in 1965 and is equivalent to

Join[{1, 2}, Map[First, CTEvolveList[{{1}, {2}}, {2}, t]]]

It is known that this sequence does not repeat, contains no more than two identical consecutive blocks, and has at least very close to equal numbers of 1's and 2's. Replacing 2 by 3 yields a sequence which has a fairly simple nested form.

Register Machines

■ **Implementation.** The state of a register machine at a particular step can be represented by the pair *{n, list}*, where *n* gives the position in the program of current instruction being executed (the "program counter") and *list* gives the values of the registers. The program for the register machine on page 99 can then be given as

```
{i[1], d[2, 1], i[2], d[1, 3], d[2, 1]}
```

where *i[_]* represents an increment instruction, and *d[_, _]* a decrement jump.

With this setup, the evolution of any register machine can be implemented using the functions (a typical initial condition is *{1, {0, 0}}*)

```
RMStep[prog_, {n_Integer, list_List}] := If[n > Length[prog],
    {n, list}, RMExecute[prog[[n]], {n, list}]]
RMExecute[i[r_], {n_, list_}] := {n + 1, MapAt[# + 1 &, list, r]}
RMExecute[d[r_, m_], {n_, list_}] :=
  If[list[[r]] > 0, {m, MapAt[# - 1 &, list, r]}, {n + 1, list}]
RMEvolveList[prog_, init : {_Integer, _List}, t_Integer] :=
  NestList[RMStep[prog, #] &, init, t]
```

The total number of possible programs of length *n* using *k* registers is $(k (1 + n))^n$. Note that by prepending suitable *i[r]* instructions one can effectively set up initial conditions with arbitrary values in registers.

■ **Halting.** It is sometimes convenient to think of register machines as going into a special halt state if they try to execute instructions beyond the end of their program. (See page 1137.) The fraction of possible register machines that do this starting from initial condition *{1, {0, 0}}* decreases steadily with program length *n*, reaching about 0.76 for *n = 8*. The most common number of steps before halting is always *n*, while the maximum numbers of steps for *n* up to 8 is *{1, 3, 5, 10, 16, 37, 215, 1280}* where in the last case this is achieved by

```
{i[1], d[2, 7], d[2, 1], i[2], i[2], d[1, 4], i[1], d[2, 3]}
```

■ **Page 101 · Extended instruction sets.** One can consider also including instructions such as

```
RMExecute[eq[r1_, r2_, m_], {n_, list_}] :=
  If[list[[r1]] == list[[r2]], {m, list}, {n + 1, list}]
RMExecute[add[r1_, r2_], {n_, list_}] :=
  {n + 1, ReplacePart[list, list[[r1]] + list[[r2]], r1]}
RMExecute[jmp[r1_], {n_, list_}] := {list[[r1]], list}
```

Note that by being able to add and subtract only 1 at each step, the register machines shown in the main text necessarily operate quite slowly: they always take at least *n* steps to build up a number of size *n*. But while extending the instruction set can increase the speed of operations, it does not appear to yield a much larger density of machines with complex behavior.

■ **History.** Register machines (also known as counter machines and program machines) are a fairly obvious idealization of practical computers, and have been invented in slightly different forms several times. Early uses of them were made by John Shepherdson and Howard Sturgis around 1959 and Marvin Minsky around 1960. Somewhat similar constructs were part of Kurt Gödel's 1931 work on representing logic within arithmetic (see page 1158).

■ **Page 102 · Random programs.** See page 1182.

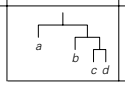
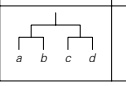
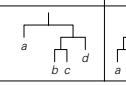
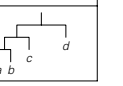
Symbolic Systems

■ **Implementation.** The evolution for *t* steps of the first symbolic system shown can be implemented simply by

```
NestList[# /. e[x_][y_] → x[x[y]] &, init, t]
```

■ **Symbolic expressions.** Expressions like *Log[x]* and *f[x]* that give values of functions are familiar from mathematics and from typical computer languages. Expressions like *f[g[x]]* giving compositions of functions are also familiar. But in general, as in *Mathematica*, it is possible to have expressions in which the head *h* in *h[x]* can itself be any expression—not just a single symbol. Thus for example *f[g][x]*, *f[g[h]][x]* and *f[g][h][x]* are all possible expressions. And these kinds of expressions often arise in *Mathematica* when one manipulates functions as a whole before applying them to arguments. ($\partial_{xx} f[x]$ for example gives *f″[x]* which is *Derivative[2][f][x]*.) (In principle one can imagine representing all objects with forms such as *f[x, y]* by so-called currying as *f[x][y]*, and indeed I tried this in the early 1980s in SMP. But although this can be convenient when *f* is a discrete function such as a matrix, it is inconsistent with general mathematical and other usage in which for example *Gamma[x]* and *Gamma[a, x]* are both treated as values of functions.)

■ **Representations.** Among the representations that can be used for expressions are:

functional	a[b[c[d]]]	a[b][c[d]]	a[b[c][d]]	a[b][c][d]
Polish	{∘, a, ∘, b, ∘, c, d}	{∘, ∘, a, b, ∘, c, d}	{∘, a, ∘, ∘, b, c, d}	{∘, ∘, ∘, a, b, c, d}
operator	a∘ (b∘ (c∘ d))	(a∘ b)∘ (c∘ d)	a∘ ((b∘ c)∘ d)	((a∘ b)∘ c)∘ d
tree	{a, {b, {c, d}}}	{{a, b}, {c, d}}	{a, {{b, c}, d}}	{{{a, b}, c}, d}
				

Typical transformation rules are non-local in all these representations. Polish representation (whose reverse form has been used in HP calculators) for an expression can be obtained using (see also page 1173)

```
Flatten[expr //. x_[y_] → {∘, x, y}]
```

The original expression can be recovered using

```
First[Reverse[list] //. {w___, x_, y_, ∘, z___} → {w, y[x], z}]
```

(Pictures of symbolic system evolution made with Polish notation differ in detail but look qualitatively similar to those made as in the main text with functional notation.)

The tree representation of an expression can be obtained using *expr //. x_[y_] → {x, y}*, and when each object has just one argument, the tree is binary, as in LISP.

If only a single symbol ever appears, then all that matters is the overall structure of an expression, which can be captured as in the main text by the sequence of opening and closing brackets, given by

```
Flatten[Characters[ToString[expr]] /.
  {"[" → 1, "]" → 0, "e" → {}}]
```

▪ **Possible expressions.** *LeafCount[expr]* gives the number of symbols that appear anywhere in an expression, while *Depth[expr]* gives the number of closing brackets at the end of its functional representation—equal to the number of levels in the rightmost branch of the tree representation. (The maximum number of levels in the tree can be computed from *expr /. _Symbol → 1 //. x_[y_] → 1 + Max[x, y]*.)

With a list *s* of possible symbols, *c[s, n]* gives all possible expressions with *LeafCount[expr] == n*:

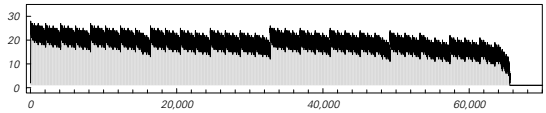
```
c[s_, 1] = s; c[s_, n_] := Flatten[
  Table[Outer[#1[#2] &, c[s, n - m], c[s, m]], {m, n - 1}]]
```

There are a total of *Binomial[2 n - 2, n - 1] Length[s]^n/n* such expressions. When *Length[s] == 1* the expressions correspond to possible balanced sequences of opening and closing brackets (see page 989).

▪ **Page 103 · Properties.** All initial conditions eventually evolve to expressions of the form *Nest[e, e, m]*, which then remain fixed. The quantity *expr //. {e → 0, x_[y_] → 2^x + y}* turns out to remain constant through the evolution, so this gives the final value of *m* for any initial condition. The maximum is *Nest[2^# &, 0, n]* (compare page 906), achieved for initial conditions of the form *Nest[#[e] &, e, n]*. (By analogy with page 1122 any *e* expression can be interpreted as a Church numeral *u = expr //. {e → 2, x_[y_] → y^x} = 2^(2^m)*, so that *expr[a][b]* evolves to *Nest[a, b, u]*.) During the evolution the rule can apply only to the inner part *FixedPoint[Replace[#, e[x_] → x] &, expr]* of an expression. The depth of this inner part for initial condition *e[e][e][e][e][e]* is shown below. For all initial conditions this depth seems at first to increase linearly, then to decrease in a nested way according to

```
FoldList[Plus, 0, Flatten[Table[
    {1, 1, Table[-1, {IntegerExponent[i, 2] + 1}]}, {i, m}]]]
```

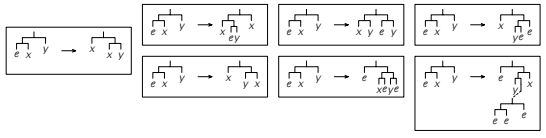
This quantity alternates between value *1* at position 2^j and value *j* at position $2^j - j + 1$. It reaches a fixed point as soon as the depth reaches 0. For initial conditions of size *n*, this occurs after at most *Sum[Nest[2^# &, 0, i] - 1, {i, n}] + 1* steps. (See also page 1145.)



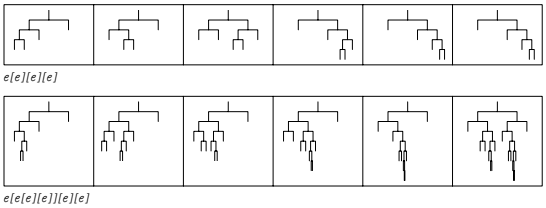
▪ **Other rules.** If only a single variable appears in the rule, then typically only nested behavior can be generated—though in an example like *e[x_][_] → e[x[e[e]][e]]* it can be quite complex. The left-hand side of each rule can consist of any expression; *e[e[x_]][y_]* and *e[e][x_[y_]]* are two possibilities. However, at least with small initial conditions it seems easier to achieve complex behavior with rules based on *e[x_][y_]*. Note that rules with no explicit *e*'s on the left-hand side always give trees with regular nested structures; *x_[y_] → x[y][x[y]]* (or *x_ → x[x]* in *Mathematica*), for example, yields balanced binary trees.

▪ **Long halting times.** Symbolic systems with rules of the form *e[x_][y_] → Nest[x, y, r]* always evolve to fixed points—though with initial conditions of size *n* this can take of order *Nest[r^# &, 0, n]* steps (see above). In general there will be symbolic systems where the number of steps to evolve to a fixed point grows arbitrarily rapidly with *n* (see page 1145), and indeed I suspect that there are even systems with quite simple rules where proving that a fixed point is always reached in a finite number of steps is beyond, for example, the axiom system for arithmetic (see page 1163).

▪ **Trees.** The rules given on pages 103 and 104 correspond to the transformations on trees shown below.



The first few steps in evolution from two initial conditions of the system on page 103 correspond to the sequences of trees below.



■ **Order dependence.** The operation *expr /. lhs → rhs* in *Mathematica* has the effect of scanning the functional representation of *expr* from left to right, and applying rules whenever possible while avoiding overlaps. (Standard evaluation in *Mathematica* is equivalent to *expr //. rules* and uses the same ordering, while *Map* uses a different order.) One can have a rule be applied only once using

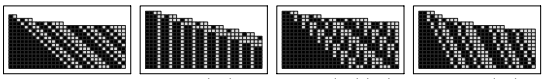
```
Module[{i = 1}, expr /. lhs :→ rhs /; i++ == 1]
```

Many symbolic systems (including the one on page 103) have the so-called Church-Rosser property (see page 1036) which implies that if a fixed point is reached in the evolution of the system, this fixed point will be the same regardless of the order in which rules are applied.

■ **History.** Symbolic systems of the general type I discuss here seem to have first arisen in 1920 in the work of Moses Schönfinkel on what became known as combinators. As discussed on page 1121 Schönfinkel introduced certain specific rules that he suggested could be used to build up functions defined in logic. Beginning in the 1930s there were a variety of theoretical studies of how logic and mathematics could be set up with combinators, notably by Haskell Curry. For the most part, however, only Schönfinkel's specific rules were ever used, and only rather specific forms of behavior were investigated. In the 1970s and 1980s there was interest in using combinators as a basis for compilation of functional programming languages, but only fairly specific situations of immediate practical relevance were considered. (Combinators have also been used as logic recreations, notably by Raymond Smullyan.)

Constructs like combinators appear to have almost never been studied in mainstream pure mathematics. Most likely the reason is that building up functions on the basis of the structure of symbolic expressions has never seemed to have much obvious correspondence to the traditional mathematical view of functions as mappings. And in fact even in mathematical logic, combinators have usually not been considered mainstream. Most likely the reason is that ever since the work of Bertrand Russell in the early 1900s it has generally been assumed that it is desirable to distinguish a hierarchy of different types of functions and objects—analogous to the different types of data supported in most programming languages. But combinators are set up not to have any restrictions associated with types. And it turns out that among programming languages *Mathematica* is almost unique in also having this same feature. And from experience with *Mathematica* it is now clear that having a symbolic system which—like combinators—has no built-in notion of types allows great generality and flexibility. (One can always set up the analog of types by having rules only for expressions whose heads have particular structures.)

■ **Operator systems.** One can generalize symbolic systems by having rules that define transformations for any *Mathematica* pattern. Often these can be thought of as one-way versions of axioms for operator systems (see page 1172), but applied only once per step (as */.* does), rather than in all possible ways (as in a multiway system)—so that the evolution is just given by *NestList[# /. rule &, init, t]*. The rule $x\_ \to x \circ x$ then for example generates a balanced binary tree. The pictures below show the patterns of opening and closing parentheses obtained from operator system evolution rules in a few cases.



■ **Network analogs.** The state of a symbolic system can always be viewed as corresponding to a tree. If a more general network is allowed then rules based on analogs of network substitution systems from page 508 can be used. (One can also construct an infinite tree from a general network by following all its possible paths, as on page 277, but in most cases there will be no simple way to apply symbolic system rules to such a tree.)

How the Discoveries in This Chapter Were Made

■ **Page 109 · Repeatability and numerical analysis.** The discrete nature of the systems that I consider in most of this book makes it almost inevitable that computer experiments on them will be perfectly repeatable. But if, as in the past, one tries to do computer experiments on continuous mathematical systems, then the situation can be different. For in such cases one must inevitably make discrete approximations for the underlying representation of numbers and for the operations that one performs on them. And in many practical situations, one relies for these approximations on "machine arithmetic"—which can differ from one computer system to another.

■ **Page 109 · Studying simple systems.** Over the years, I have watched with disappointment the continuing failure of most scientists and mathematicians to grasp the idea of doing computer experiments on the simplest possible systems. Those with physical science backgrounds tend to add features to their systems in an attempt to produce some kind of presumed realism. And those with mathematical backgrounds tend to add features to make their systems fit in with complicated and abstract ideas—often related to continuity—that exist in modern mathematics. The result of all this has been that remarkably few truly meaningful computer experiments have ended up ever being done.

▪ **Page 111 · The relevance of theorems.** Following traditional mathematical thinking, one might imagine that the best way to be certain about what could possibly happen in some particular system would be to prove a theorem about it. But in my experience, proofs tend to be subject to many of the same kinds of problems as computer experiments: it is easy to end up making implicit assumptions that can be violated by circumstances one cannot foresee. And indeed, by now I have come to trust the correctness of conclusions based on simple systematic computer experiments much more than I trust all but the simplest proofs.

▪ **Attitudes of mathematicians.** Mathematicians often seem to feel that computer experimentation is somehow less precise than their standard mathematical methods. It is true that in studying questions related to continuous mathematics, imprecise numerical approximations have often been made when computers are used (see above). But discrete or symbolic computations can be absolutely precise. And in a sense presenting a particular object found by experiment (such as a cellular automaton whose evolution shows some particular property) can be viewed as a constructive existence proof for such an object. In doing mathematics there is often the idea that proofs should explain the result they prove—and one might not think this could be achieved if one just presents an object with certain properties. But being able to look in detail at how such an object works will in many cases provide a much better understanding than a standard abstract mathematical proof. And inevitably it is much easier to find new results by the experimental approach than by the traditional approach based on proofs.

▪ **History of experimental mathematics.** The general idea of finding mathematical results by doing computational experiments has a distinguished, if not widely discussed, history. The method was extensively used, for example, by Carl Friedrich Gauss in the 1800s in his studies of number theory, and presumably by Srinivasa Ramanujan in the early 1900s in coming up with many algebraic identities. The Gibbs phenomenon in Fourier analysis was noticed in 1898 on a mechanical computer constructed by Albert Michelson. Solitons were rediscovered in experiments done around 1954 on an early electronic computer by Enrico Fermi and collaborators. (They had been seen in physical systems by John Scott Russell in 1834, but had not been widely investigated.) The chaos phenomenon was noted in a computer experiment by Edward Lorenz in 1962 (see page 971). Universal behavior in iterated maps (see page 921) was discovered by Mitchell Feigenbaum in 1975 by looking at examples from an electronic calculator. Many aspects of fractals were found by Benoit Mandelbrot in the 1970s using computer graphics. In the 1960s and 1970s a variety of algebraic identities were found using computer algebra, notably by William Gosper. (Starting in the mid-1970s I routinely did computer algebra experiments to find formulas in theoretical physics—though I did not mention this when presenting the formulas.) The idea that as a matter of principle there should be truths in mathematics that can only be reached by some form of inductive reasoning—like in natural science—was discussed by Kurt Gödel in the 1940s and by Gregory Chaitin in the 1970s. But it received little attention. With the release of *Mathematica* in 1988, mathematical experiments began to emerge as a standard element of practical mathematical pedagogy, and gradually also as an approach to be tried in at least some types of mathematical research, especially ones close to number theory. But even now, unlike essentially all other branches of science, mainstream mathematics continues to be entirely dominated by theoretical rather than experimental methods. And even when experiments are done, their purpose is essentially always just to provide another way to look at traditional questions in traditional mathematical systems. What I do in this book—and started in the early 1980s—is, however, rather different: I use computer experiments to look at questions and systems that can be viewed as having a mathematical character, yet have never in the past been considered in any way by traditional mathematics.

▪ **Page 113 · Practicalities.** The investigations described in this chapter were done using *Mathematica*, mostly in 1992. For larger searches, I sometimes created optimized C programs that were controlled via *MathLink* from within *Mathematica*—though with the versions of *Mathematica* that exist today this would now be unnecessary. For my very largest searches, I used *Mathematica* to dispatch programs to a large number of different computers on a network, then had the computers send me email whenever they found interesting results. (See also page 854.)

Systems Based on Numbers

The Notion of Numbers

■ **Implementation of digit sequences.** A whole number *n* can be converted to a sequence of digits in base *k* using *IntegerDigits[n, k]* or (see also page 1094)

```
Reverse[Mod[NestWhileList[Floor[#/k] &, n, # ≥ k &], k]]
```

and from a sequence of digits using *FromDigits[list, k]* or

```
Fold[k #1 + #2 &, 0, list]
```

For a number *x* between 0 and 1, the first *m* digits in its digit sequence in base *k* are given by *RealDigits[x, k, m]* or

```
Floor[k NestList[Mod[k #, 1] &, x, m - 1]]
```

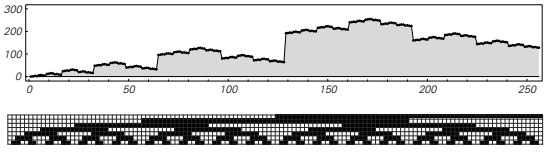
and from these digits one can reconstruct an approximation to the number using *FromDigits[{list, 0}, k]* or

```
Fold[#1/k + #2 &, 0, Reverse[list]]/k
```

■ **Gray code.** In looking at digit sequences, it is sometimes useful to consider ordering numbers by a criterion other than their size. An example is Gray code ordering, in which successive numbers are arranged to differ in only one digit. One possible such ordering for numbers with a total of *m* digits is

```
GrayCode[m_] :=
  Nest[Join[#, Length[#] + Reverse[#]] &, {0}, m]
```

The succession of sizes and digit sequences of numbers ordered in this way are shown below. (Note that the digit sequence picture is turned on its side relative to those in the main text). The number which appears at position *i* is given by *BitXor[i, Floor[i/2]]*. (Iterating the related function *BitXor[i, 2 i]* yields numbers whose digit sequences correspond to the rule 60 cellular automaton).



■ **A note for mathematicians.** Some mathematicians will at first find what I say in this chapter quite bizarre. It may help however to point out that the traditional view of numbers already shows signs of breaking down in many studies of dynamical systems done over the past few decades. Thus for example, instead of getting results in terms of continuous functions, Cantor sets very often appear. Indeed, the symbolic dynamics approach that is often used in dynamical systems theory is quite close to the digit sequence approach I use here—Markov partitions in dynamical systems theory are essentially just generalizations of digit expansions.

However, in the cases that are analyzed in dynamical systems theory, only shifts and other very simple operations are typically performed on digit sequences. And as a result, most of the phenomena that I discuss in this chapter have not been seen in work done in dynamical systems theory.

■ **History of numbers.** Numbers were probably first used many thousands of years ago in commerce, and initially only whole numbers and perhaps rational numbers were needed. But already in Babylonian times, practical problems of geometry began to require square roots. Nevertheless, for a very long time, and despite some development of algebra, only numbers that could somehow in principle be constructed mechanically were ever considered. The invention of fluxions by Isaac Newton in the late 1600s, however, introduced the idea of continuous variables—numbers with a continuous range of possible sizes. But while this was a convenient and powerful notion, it also involved a new level of abstraction, and it brought with it considerable confusion about fundamental issues. In fact, it was really only through the development of rigorous mathematical analysis in the late 1800s that this confusion finally began to clear up. And already by the 1880s Georg Cantor and others had constructed completely discontinuous functions, in which the idea of treating numbers as continuous variables where only the size matters was called into question. But until almost the 1970s, and the emergence of fractal geometry and chaos theory, these functions were largely considered as

mathematical curiosities, of no practical relevance. (See also page 1168.)

Independent of pure mathematics, however, practical applications of numbers have always had to go beyond the abstract idealization of continuous variables. For whether one does calculations by hand, by mechanical calculator or by electronic computer, one always needs an explicit representation for numbers, typically in terms of a sequence of digits of a certain length. (From the 1930s to 1960s, some work was done on so-called analog computers which used electrical voltages to represent continuous variables, but such machines turned out not to be reliable enough for most practical purposes.) From the earliest days of electronic computing, however, great efforts were made to try to approximate a continuum of numbers as closely as possible. And indeed for studying systems with fairly simple behavior, such approximations can typically be made to work. But as we shall see later in this chapter, with more complex behavior, it is almost inevitable that the approximation breaks down, and there is no choice but to look at the explicit representations of numbers. (See also page 1128.)

■ **History of digit sequences.** On an abacus or similar device numbers are in effect represented by digit sequences. In antiquity however most systems for writing numbers were like the Roman one and not based on digit sequences. An exception was the Babylonian base 60 system (from which hours:minutes:seconds notation derives). The Hindu-Arabic base 10 system in its modern form probably originated around 600 AD, and particularly following the work of Leonardo Fibonacci in the early 1200s, became common by the 1400s. Base 2 appears to have first been considered explicitly in the early 1600s (notably by John Napier in 1617), and was studied in detail by Gottfried Leibniz starting in 1679. The possibility of arbitrary bases was stated by Blaise Pascal in 1658. Various bases were used in puzzles, but rarely in pure mathematics (work by Georg Cantor in the 1860s being an exception). The first widespread use of base 2 was in electronic computers, starting in the late 1940s. Even in the 1980s digit sequences were viewed by most mathematicians as largely irrelevant for pure mathematical purposes. The study of fractals and nesting, the appearance of many algorithms involving digit sequences and the routine use of long numbers in *Mathematica* have however gradually made digit sequences be seen as more central to mathematics.

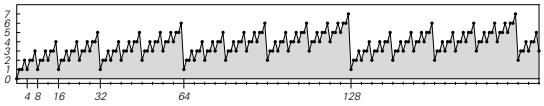
Elementary Arithmetic

■ **Page 117 · Substitution systems.** There are many connections between digit sequences and substitution systems, as discussed on page 891. The pattern shown here is essentially a rotated version of the pattern generated by the first substitution system on page 83.

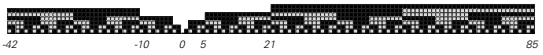
■ **Page 117 · Digit counts.** The number of black squares on row n in the pattern shown here is given by *DigitCount[n, 2, 1]* and is plotted below. This function appeared on page 870 in the discussion of binomial coefficients modulo 2, and will appear again in several other places in this book. Note the inequality *1 ≤ DigitCount[n, 2, 1] ≤ Log[2, n]*. Formulas for *DigitCount[n, 2, 1]* include *n - IntegerExponent[n!, 2]* and

2 n - Log[2, Denominator[Derivative[n][(1 - #)^-1/2 &][0]/n!]]

Straightforward generalizations of *DigitCount* can be defined for integer and non-integer bases and by looking not only at the total number of digits but also at correlations between digits. In all cases the analogs of the picture below have a nested structure.

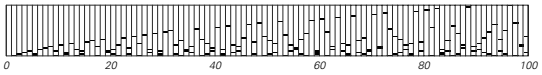


■ **Negative bases.** Given a suitable list of digits from 0 to $k - 1$ one can obtain any positive or negative number using *FromDigits[list, -k]*. The picture below shows the digit sequences of successive numbers in base -2; the row j from the bottom turns out to consist of alternating black and white blocks of length 2^j. (In ordinary base 2 a number $-n$ can be represented as on a typical electronic computer by complementing each digit, including leading 0's.) (See also page 1093.)

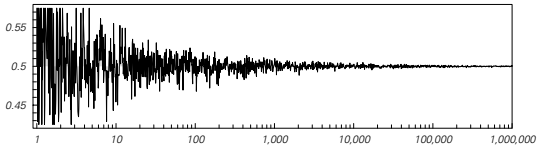


■ **Non-power bases.** One can consider representing numbers by *Sum[a[n] f[n], {n, 0, ∞}]* where the *f[n]* need not be k^n. So long as *f[n]* grows less rapidly than 2^n (as when *f = Fibonacci* or *f = Prime*), digits 0 and 1 will suffice, though the representation is not generally unique. (See page 1070.)

■ **Multiplicative digit sequences.** One can consider generalizations of digit sequences in which numbers are broken into parts combined not by addition but by multiplication. Since numbers can be factored uniquely into products of powers of primes, a number can be specified by a list in which 1's appear at the positions of the appropriate *Prime[m]*^{n} (which can be sorted by size) and 0's appear elsewhere, as shown below. Note that unlike the case of ordinary additive digits, far more than *Log[m]* digits are required to specify a number m.



▪ **Page 120 · Powers of three in base 2.** The n^{th} row in the pattern shown can be obtained simply as *IntegerDigits[3^n, 2]*. Even such individual rows seem in many respects random. The picture below shows the fraction of 1's that appear on successive rows. The fraction seems to tend to 1/2.



If one looks only at the rightmost s columns of the pattern, one sees repetition—but the period of the repetition grows like 2^s. Typical vertical columns have one obvious deviation from randomness: it is twice as probable for the same colors to occur on successive steps than for opposite colors. (For multiplier m in base k, the relative frequencies of pairs *{i, j}* are given by *Quotient[a i - j - 1 + m, k] - Quotient[m i - j - 1, k]*.)

The sequence *Mod[3^n, 2^s]* obtained from the rightmost s digits corresponds to a simple linear congruential pseudorandom number generator. Such generators are widely used in practical computer systems, as discussed further on page 974. (Note that in the particular case used here, pairs of numbers *Mod[{3^n, 3^{n+1}}, 2^s]* always lie on lines; with multipliers other than 3, such regularities may occur for longer blocks of numbers.)

Note that if one uses base 6 rather than base 2, then as shown on page 614 powers of 3 still yield a complicated pattern, but all operations are strictly local, and the system corresponds to a cellular automaton with 6 possible colors for each cell and rule *{a_, b_, c_} → 3 Mod[b, 2] + Floor[c/2]* (see page 1093).

▪ **Leading digits.** In base b the leading digits of powers are not equally probable, but follow the logarithmic law from page 914.

▪ **Page 122 · Powers of 3/2.** The n^{th} value shown in the plot here is *Mod[(3/2)^n, 1]*. Measurements suggest that these values are uniformly distributed in the range 0 to 1, but despite a fair amount of mathematical work since the 1940s, there has been no substantial progress towards proving this.

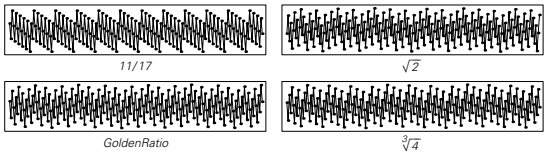
In base 6, $(3/2)^n$ is a cellular automaton with rule

```
{a_, b_, c_} → 3 Mod[a + Quotient[b, 2], 2] +
    Quotient[3 Mod[b, 2] + Quotient[c, 2], 2]
```

(Note that this rule is invertible.) Looking at $u\,(3/2)^n$ then corresponds to studying the cellular automaton with an initial condition given by the base 6 digits of u. It is then possible to find special values of u (an example is 0.166669170371...) which make the first digit in the fractional part of $u\,(3/2)^n$ always nonzero, so that *Mod[u (3/2)^n, 1] > 1/6*. In general, it seems that *Mod[u (3/2)^n, 1]* can be kept as large as about *0.3* (e.g. with *u = 0.38906669065 ...*) but no larger.

▪ **General powers.** It has been known in principle since the 1930s that *Mod[h^n, 1]* is uniformly distributed in the range 0 to 1 for almost all values of h. However, no specific value of h for which this is true has ever been explicitly found. (Some attempts to construct such values were made in the 1970s.) Exceptions are known to include so-called Pisot numbers such as *GoldenRatio*, $\sqrt{2}+1$ and *Root[#^3 - # - 1 &, 1]* (the numerically smallest of all Pisot numbers) for which *Mod[h^n, 1]* becomes 0 or 1 for large n. Note that *Mod[x h^n, 1]* effectively extracts successive digits of x in base h (see pages 149 and 919).

▪ **Multiples of irrational numbers.** Instead of powers one can consider successive multiples *Mod[h n, 1]* of a number h. The pictures below show results obtained as a function of n for various choices of h. (These correspond to positions of a particle bouncing around in an idealized box, as discussed on pages 971 and 1022.)



When h is a rational number, the sequence always repeats. But in all other cases, the sequence does not repeat, and in fact it is known that a uniform distribution of values is obtained. (The average difference of successive values is maximized for *h = GoldenRatio*, as mentioned on page 891.)

▪ **Relation to substitution systems.** Despite the uniform distribution result in the note above, the sequence *Floor[(n + 1) h] - Floor[n h]* is definitely not completely random, and can in fact be generated by a sequence of substitution rules. The first m rules (which yield far more than m elements of the original sequence) are obtained for any h that is not a rational number from the continued fraction form (see page 914) of h by

```
Map[({0 → Join[#, {1}], 1 → Join[#, {1, 0}]} &)[Table[0,
    {# - 1}]] &, Reverse[Rest[ContinuedFraction[h, m]]]]
```

Given these rules, the original sequence is given by

```
Floor[h] + Fold[Flatten[#1 /. #2] &, {0}, rules]
```

If h is the solution to a quadratic equation, then the continued fraction form is repetitive, and so there are a limited number

of different substitution rules. In this case, therefore, the original sequence can be found by a neighbor-independent substitution system of the kind discussed on page 82. For *h = GoldenRatio* the substitution system is *{0 → {1}, 1 → {1, 0}}* (see page 890), for $h = \sqrt{2}$ it is *{0 → {0, 1}, 1 → {0, 1, 0}}* (see page 892) and for $h = \sqrt{3}$ it is *{0 → {1, 1, 0}, 1 → {1, 1, 0, 1}}*. (The presence of nested structure is particularly evident in *FoldList[Plus, 0, Table[Mod[h n, 1] - 1/2, {n, max}]]*.) (See also pages 892, 916, 932 and 1084.)

■ **Other uniformly distributed sequences.** Cases in which *Mod[a[n], 1]* is uniformly distributed include $\sqrt{n}$, *n Log[n]*, *Log[Fibonacci[n]]*, *Log[n!]*, $h n^2$ and *h Prime[n]* (*h* irrational) and probably *n Sin[n]*. (See also page 914.)

■ **Page 122 · Implementation.** The evolution for *t* steps of the system at the top of the page can be computed simply by

```
NestList[If[EvenQ[#], 3#/2, 3(# + 1)/2] &, 1, t]
```

■ **Page 122 · The *3n+1* problem.** The system described here is similar to the so-called *3 n + 1* problem, in which one looks at the rule *n → If[EvenQ[n], n/2, (3 n + 1)/2]* and asks whether for any initial value of *n* the system eventually evolves to 1 (and thereafter simply repeats the sequence 1, 2, 1, 2, ...). It has been observed that this happens for all initial values of *n* up to at least 10^{16}, but despite a fair amount of mathematical effort since the problem was first posed in the 1930s, no general proof for all values of *n* has ever been found. (For negative initial *n*, the evolution appears always to reach -1, -5 or -17, and then repeat with periods 1, 3 or 11 respectively.) An alternative formulation is to ask whether for all *n*

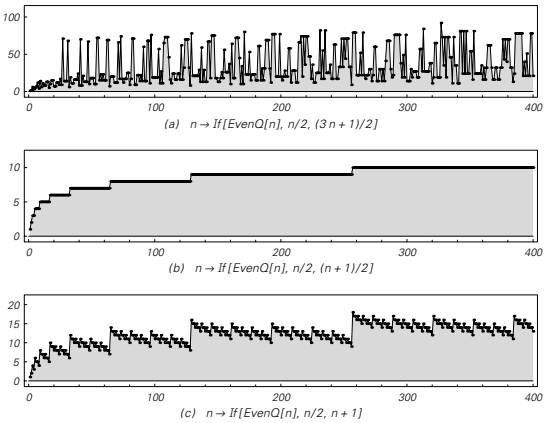
```
FixedPoint[(3#/2^IntegerExponent[#, 2] + 1)/2 &, n] == 2
```

With the rule *n → If[EvenQ[n], 5 n/2, (n + 1)/2]* used in the main text, the sequence produced repeats if *n* ever reaches 2, 4 or 40 (and possibly higher numbers). But with initial values of *n* up to 10,000, this happens in only 642 cases, and with values up to 100,000 it happens in only 2683 cases. In all other cases, the values of *n* in the sequence appear to grow forever.

To get some idea about the origin of this behavior, one can assume that successive values of *n* are randomly even and odd with equal probability. And with this assumption, *n* should increase by a factor of 5/2 half the time, and decrease by a factor close to 1/2 the rest of the time—so that after *t* steps it should be multiplied by an overall factor of about $(\sqrt{5}/2)^t$. Starting with *n = 6*, the effective exponents for *t = 10^Range[6]* are *{39.6, 245.1, 1202.8, 9250.7, 98269.8, 1002020.4}*. One reason that all sequences do not grow forever is that even with perfect randomness, there will be fluctuations, and occasionally *n* will reach a low value that makes it get stuck in a repetitive sequence.

If one applies the same kind of argument to the standard *3 n + 1* problem, then one concludes that *n* should on average decrease by a factor of $\sqrt{3}/2$ at each step, making it unsurprising that at least in most cases *n* eventually reaches the value 1. Indeed, averaging over many initial values of *n*, there is good quantitative agreement between the predictions of the randomness approximation and the actual *3 n + 1* problem. But since there is no fundamental basis for the randomness approximation, it is still conceivable that a particular value of *n* exists that does not follow its predictions.

The pictures below show how many steps are needed to reach value 1 starting from different values of *n*. Case (a) is the standard *3 n + 1* problem. Cases (b) and (c) use somewhat different rules that yield considerably simpler behavior. In case (b), the number of steps is equal to the number of base 2 digits in *n*, while in case (c) it is determined by the number of 1's in the base 2 digit sequence of *n*.



■ ***3n+1* problem as cellular automaton.** If one writes the digits of *n* in base 6, then the rule for updating the digit sequence is a cellular automaton with 7 possible colors (color 6 works as an end marker that appears to the left and right of the actual digit sequence):

```
{a_, b_, c_} → If[b == 6, If[EvenQ[a], 6, 4],
    3 Mod[a, 2] + Quotient[b, 2] /. 0 :> 6 /; a == 6]
```

The *3 n + 1* problem can then be viewed as a question about the existence of persistent structure in this cellular automaton.

■ **Reconstructing initial conditions.** Given a particular starting value of *n*, it is difficult to predict what precise sequence of even and odd values will be obtained in the system on page 122. But given *t* steps in this sequence as a list of 0's and 1's, the

following function will reconstruct the rightmost t digits in the starting value of n:

```
IntegerDigits[First[Fold[{Mod[If[OddQ[#2], 2 First[#1] - 1,
      2 First[#1] PowerMod[5, -1, Last[#1]]], Last[#1]],
    2 Last[#1]} &, {0, 2}, Reverse[list]]], 2, Length[list]]
```

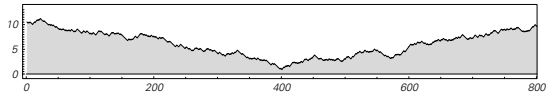
■ **A reversible system.** In both the ordinary $3n+1$ problem and in the systems discussed in the main text different numbers often evolve to the same value so that there is no unique way to reverse the evolution. However, with the rule

```
n → If[EvenQ[n], 3 n/2, Round[3 n/4]]
```

it is always possible to go backwards by the rule

```
n → If[Mod[n, 3] == 0, 2 n/3, Round[4 n/3]]
```

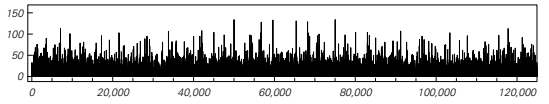
The picture shows the number of base 10 digits in numbers obtained by backward and forward evolution from $n = 8$. For $n < 8$, the system always enters a short cycle. Starting at $n = 44$, there is also a length 12 cycle. But apart from these cycles, the numbers produced always seem to grow without bound at an average rate of $3/(2\sqrt{2})$ in the forward direction, and $2\,4^{1/3}/3$ in the backward direction (at least all numbers up to 10,000 grow to above 10^{100}). Approximately one number in 20 has the property that evolution either backward or forward from it never leads to a smaller number.



■ **Page 125 · Reversal-addition systems.** The operation that is performed here is

```
n → n + FromDigits[Reverse[IntegerDigits[n, 2]], 2]
```

After a few steps, the digit sequence obtained is typically reversal symmetric (a generalized palindrome) except for the interchange of 0 and 1, and for the presence of localized structures. The sequence expands by at least one digit every two steps; more rapid expansion is typically correlated with increased randomness. For most initial n, the overall pattern obtained quickly becomes repetitive, with an effective period of 4 steps. But with the initial condition $n = 512$, no repetition occurs for at least a million steps, at which point n has 568418 base 2 digits. The plot below shows the lengths of the successive regions of regularity visible on the right-hand edge of the picture on page 126 over the course of the first million steps.



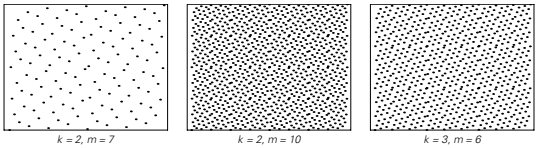
If one works directly with a digit sequence of fixed length, dropping any carries on the left, then a repetitive pattern is typically obtained fairly quickly. If one always includes one new digit on the left at every step, even when it is 0, then a rather random pattern is produced.

■ **History.** Systems similar to the one described here (though often in base 10) were mentioned in the recreational mathematics literature at least as long ago as 1939. A few small computer experiments were done around 1970, but no large-scale investigations seem to have previously been made.

■ **Digit reversal.** Sequences of the form

```
Table[FromDigits[
    Reverse[IntegerDigits[n, k, m]], k], {n, 0, k^m - 1}]
```

shown below appear in algorithms such as the fast Fourier transform and, with different values of k for different coordinates, in certain quasi-Monte Carlo schemes. (See pages 1073 and 1085.) Such sequences were considered by Johannes van der Corput in 1935.



■ **Iterated run-length encoding.** Starting say with $\{1\}$ consider repeatedly replacing *list* by (see page 1070)

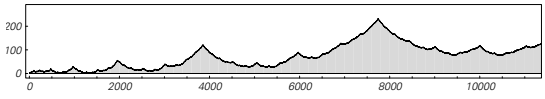
```
Flatten[Map[{Length[#], First[#]} &, Split[list]]]
```

The resulting sequences contain only the numbers 1, 2 and 3, but otherwise at first appear fairly random. However, as noticed by John Conway around 1986, the sequences can actually be obtained by a neighbor-independent substitution system, acting on 92 subsequences, with rules such as $\{3, 1, 1, 3, 3, 2, 2, 1, 1, 3\} \to \{\{1, 3, 2\}, \{1, 2, 3, 2, 2, 2, 1, 1, 3\}\}$. The system thus in the end produces patterns that are purely nested, though formed from rather complicated elements. The length of the sequence at the n^{th} step grows like λ^n, where $\lambda \simeq 1.3$ is the root of a degree 71 polynomial, corresponding to the largest eigenvalue of the transition matrix for the substitution system.

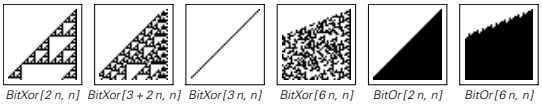
■ **Digit count sequences.** Starting say with $\{1\}$ repeatedly replace *list* by

```
Join[list, IntegerDigits[Apply[Plus, list], 2]]
```

The resulting sequences grow in length roughly like $n\,\text{Log}[n]$. The picture below shows the fluctuations around $m/2$ of the cumulative number of 1's up to position m in the sequence obtained at step 1000. A definite nested structure similar to picture (c) on page 130 is evident.



■ **Iterated bitwise operations.** The pictures below show digit sequences generated by repeatedly applying combinations of bitwise and arithmetic operations. The first example corresponds to elementary cellular automaton rule 60. Note that any cellular automaton rule can be reproduced by some appropriate combination of bitwise and arithmetic operations.



BitXor[2 n, n] BitXor[3 + 2 n, n] BitXor[3 n, n] BitXor[6 n, n] BitOr[2 n, n] BitOr[6 n, n]

Recursive Sequences

■ **Page 128 · Recurrence relations.** The rules for the sequences given here all have the form of linear recurrence relations. An explicit formula for the n^{th} term in each sequence can be found by solving the algebraic equation obtained by applying the replacement $f[m\_] \to t^m$ to the recurrence relation. (In case (e), for example, the equation is $t^n == -t^{n-1} + t^{n-2}$.) Note that (d) is the Fibonacci sequence, discussed on page 890.

Standard examples of recursive sequences that do not come from linear recurrence relations include factorial

```
f[1] = 1; f[n_] := n f[n - 1]
```

and Ackermann functions (see below). These two sequences both grow rapidly, but smoothly.

A recurrence relation like

```
f[0] = x; f[n_] := a f[n - 1] (1 - f[n - 1])
```

corresponds to an iterated map of the kind discussed on page 920, and has complicated behavior for many rational x.

■ **Ackermann functions.** A convenient example is

```
f[1, n_] := n; f[m_, 1] := f[m - 1, 2]

f[m_, n_] := f[m - 1, f[m, n - 1] + 1]
```

The original function constructed by Wilhelm Ackermann around 1926 is essentially

```
f[1, x_, y_] := x + y;
f[m_, x_, y_] := Nest[f[m - 1, x, #] &, x, y - 1]
```

or

```
f[m_, x_, y_] :=
  Nest[Function[z, Nest[#1, x, z - 1]] &, x + # &, m - 1][y]
```

For successive m (following the so-called Grzegorczyk hierarchy) this is $x + y$, $x\,y$, x^y, $\text{Nest}[x^{\#}\,\&, 1, y]$, ... `f[4, x, y]` can also be written `Array[x &, y, 1, Power]` and is sometimes called tetration and denoted $x\uparrow\uparrow y$.

■ **Page 129 · Computation of sequences.** It is straightforward to compute the various sequences given here, but to avoid a rapid increase in computer time, it is essential to store all the values of `f[n]` that one has already computed, rather than recomputing them every time they are needed. This is achieved for example by the definitions

```
f[n_] := f[n] = f[n - f[n - 1]] + f[n - f[n - 2]]

f[1] = f[2] = 1
```

The question of which recursive definitions yield meaningful sequences can depend on the details of how the rules are applied. For example, `f[-1]` may occur, but if the complete expression is `f[-1] - f[-1]`, then the actual value of `f[-1]` is irrelevant. The default form of evaluation for recursive functions implemented by all standard computer languages (including *Mathematica*) is the so-called leftmost innermost scheme, which attempts to find explicit values for each `f[k]` that occurs first, and will therefore never notice if `f[k]` in fact occurs only in the combination `f[k] - f[k]`. (The SMP system that I built around 1980 allowed different schemes—but they rarely seemed useful and were difficult to understand.)

■ **Page 131 · Properties of sequences.** Sequence (d) is given by

```
f[n_] := (n + g[IntegerDigits[n, 2]])/2

g[{(1)..}] = 1; g[{1, (0)..}] = 0

g[{1, s__}] := 1 + g[IntegerDigits[FromDigits[{s}, 2] + 1, 2]]
```

The list of elements in the sequence up to value m is given by

```
Flatten[Table[Table[n, {IntegerExponent[n, 2] + 1}], {n, m}]]
```

The differences between the first $2(2^k - 1)$ of these elements is

```
Nest[Replace[#, {x___} → {x, 1, x, 0}] &, {}, k]
```

The largest n for which `f[n] == m` is given by $2m + 1 - \text{DigitCount}[m, 2, 1]$ or $\text{IntegerExponent}[(2m)!, 2] + 1$ (this satisfies `h[1] = 2; h[m_] := h[Floor[m/2]] + m`).

The form of sequence (c) is similar to that obtained from concatenation numbers on page 913. Hump m in the picture of sequence (c) shown is given by

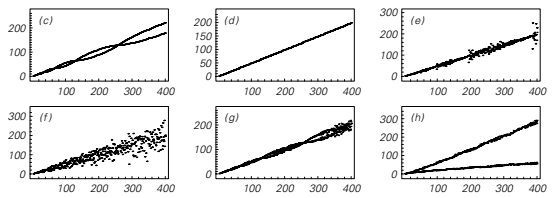
```
FoldList[Plus, 0, Flatten[Nest[Delete[NestList[Rest, #,
  Length[#] - 1], 2] &, Append[Table[1, {m}], 0], m]] - 1/2]
```

The first 2^m elements in the sequence can also be generated in terms of reordered base 2 digit sequences by

```
FoldList[Plus, 1, Map[Last[Last[#]] &,
    Sort[Table[({Length[#], Apply[Plus, #], 1 - #} &)[
        IntegerDigits[i, 2]], {i, 2^m}]]]]
```

Note that the positive and negative fluctuations in sequence (f) are not completely random: although the probability for individual fluctuations in each direction seems to be the same, the probability for two positive fluctuations in a row is smaller than for two negative fluctuations in a row.

In the sequences discussed here, `f[n_]` always has the form `f[p[n]] + f[q[n]]`. The plots at the top of the next page show `p[n]` and `q[n]` as a function of n.



The process of evaluating *f[n]* for a particular *n* can be thought of as yielding a tree where each node is a particular *f[k]* which has two successors, *f[p[k]]* and *f[q[k]]*. The distinct nodes reached starting from *f[12]* for sequence (f) are then for example *{{12}, {3, 7}, {1, 2, 4}, {1, 2}, {1}}*. The total lengths of these chains (corresponding to the depth of the evaluation tree) seem to increase roughly like *Log[n]* for all the rules on this page. For the Fibonacci sequence, it is instead *n – 1*. The maximum number of distinct nodes at any level in the tree has large fluctuations but its peaks seem to increase roughly linearly for all the rules on this page (in the Fibonacci case it is *Ceiling[n/2]*).

■ **History.** The idea of sequences in which later terms are deduced from earlier ones existed in antiquity, notably in the method of induction and in various approximation schemes (compare page 918). The Fibonacci sequence also appears to have arisen in antiquity (see page 890). A fairly clear idea of integer recurrence relations has existed since about the 1600s, but until very recently mainstream mathematics has almost never investigated them. In the late 1800s and early 1900s issues about the foundations of mathematics (see note below) led to the formal definition of so-called recursive functions. But almost without exception the emphasis was on studying what such functions could in principle do, not on looking at the actual behavior of particular ones. And indeed, despite their simple forms, recursive sequences of the kind I discuss here do not for the most part ever appear to have been studied before—although sequence (c) was mentioned in lectures by John Conway around 1988, and the first 17 terms of sequence (e) were given by Douglas Hofstadter in 1979.

■ **Primitive recursive functions.** As part of trying to formalize foundations of arithmetic Richard Dedekind began around 1888 to discuss possible functions that could be defined using recursion (induction). By the 1920s there had then emerged a definite notion of primitive recursive functions. The proof of Gödel's Theorem in 1931 made use of both primitive and general recursive functions—and by the mid-1930s emphasis had shifted to discussion of general recursive functions.

Primitive recursive functions are defined to deal with non-negative integers and to be set up by combining the basic functions *z = 0 &* (zero), *s = # + 1 &* (successor) and *p[i_] := Slot[i] &* (projection) using the operations of composition and primitive recursion

```
f[0, y___Integer] := g[y]
f[x_Integer, y___Integer] := h[f[x - 1, y], x - 1, y]
```

Plus and *Times* can then for example be defined as

```
plus[0, y_] = y; plus[x_, y_] := s[plus[x - 1, y]]
times[0, y_] = 0; times[x_, y_] := plus[times[x - 1, y], y]
```

Most familiar integer mathematical functions also turn out to be primitive recursive—examples being *Power*, *Mod*, *Binomial*, *GCD* and *Prime*. And indeed in the early 1900s it was thought that perhaps any function that could reasonably be computed would be primitive recursive (see page 1125). But the construction in the late 1920s of the Ackermann function *f[m, x, y]* discussed above showed that this was not correct. For any primitive recursive function can grow for large *x* at most like *f[m, x, x]* with fixed *m*. Yet *f[x, x, x]* will always eventually grow faster than this—demonstrating that the whole Ackermann function cannot be primitive recursive. (See page 1162.)

A crucial feature of primitive recursive functions is that the number of steps they take to evaluate is always limited, and can always in effect be determined in advance, since the basic operation of primitive recursion can be unwound simply as

```
f[x_, y___] := Fold[h[#1, #2, y] &, g[y], Range[0, x - 1]]
```

And what this means is that any computation that for example fundamentally involves a search that might not terminate cannot be implemented by a primitive recursive function. General recursive functions, however, also allow

```
μ[f_] = NestWhile[# + 1 &, 0, Function[n, f[n, ##1] ≠ 0]] &
```

which can perform unbounded searches. (Ordinary primitive recursive functions are always total functions, that give definite values for every possible input. But general recursive functions can be partial functions, that do not terminate for some inputs.) As discussed on page 1121 it turns out that general recursive functions are universal, so that they can be used to represent any possible computable function. (Note that any general recursive function can be expressed in the form *c[f, μ[g]]* where *f* and *g* are primitive recursive.)

In enumerating recursive functions it is convenient to use symbolic definitions for composition and primitive recursion

```
c[g_, h___] = Apply[g, Through[{h}[##]]] &
r[g_, h_] =
  If[#1 == 0, g[##2], h[#0[#1 - 1, ##2], #1 - 1, ##2]] &
```

where the more efficient unwound form is

```
r[g_, h_] = Fold[Function[{u, v}, h[u, v, ##2]],
                 g[##2], Range[0, # - 1]] &
```

And in terms of these, for example, *plus = r[p[1], s]*.

The total number of recursive functions grows roughly exponentially in the size (*LeafCount*) of such expressions, and roughly linearly in the number of arguments.

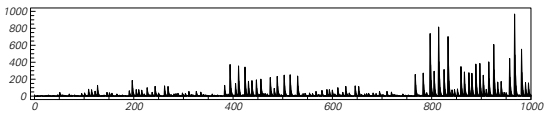
Most randomly selected primitive recursive functions show very simple behavior—either constant or linearly increasing when fed successive integers as arguments. The smallest examples that show other behavior are:

- *r[z, r[s, s]]*, which is *1/2 # (# + 1) &*, with quadratic growth
- *r[z, r[s, c[s, s]]]*, which is $2^{\#+1} - \# - 2$ &, with exponential growth
- *r[z, r[s, p[2]]]*, which is *2^Ceiling[Log[2, # + 2]] - # - 2 &*, which shows very simple nesting
- *r[z, r[c[s, z], z]]*, which is *Mod[#, 2] &*, with repetitive behavior
- *r[z, r[s, r[s, s]]]* which is *Fold[1/2 #1 (#1 + 1) + #2 &, 0, Range[#]] &*, growing like 2^{2^x}.

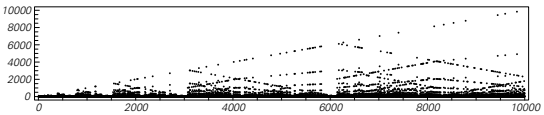
r[z, r[s, r[s, r[s, p[2]]]]] is the first function to show significantly more complex behavior, and indeed as the picture below indicates, it already shows remarkable randomness. From its definition, the function can be written as

```
Fold[Fold[2^Ceiling[Log[2, Ceiling[(#1 + 2)/(#2 + 2)]]]
    (#2 + 2) - 2 - #1 &, #2, Range[#1]] &, 0, Range[#]] &
```

Its first zeros are at *{4, 126, 813, 966, 1166, 1177, 1666, 1897}*.



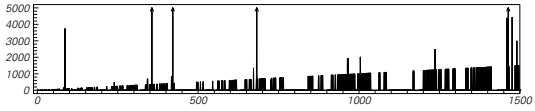
Each zero is immediately followed by a maximum equal to x, and as picture below shows, values tend to accumulate for example on lines of the form $\pm x/2^u \pm (2m+1)2^v$.



Note that functions of the form *Nest[r[c[s, z], #] &, c[s, s], n]* are given in terms of the original Ackermann function in the note above by *f[n + 1, 2, # + 1] - 1 &*.

Before the example above one might have thought that primitive recursive functions would always have to show rather simple behavior. But already an immediate counterexample is *Prime*. And it turns out that if they never sample values below *f[0]* the functions in the main text are also all primitive recursive. (Their definitions have a primitive recursive structure, but to operate correctly they must be given integers that are non-negative.)

Among functions with simple explicit definitions, essentially the only examples known fundamentally to be not primitive recursive are ones closely related to the Ackermann function. But given an enumeration of primitive recursive functions (say ordered first by *LeafCount*, then with *Sort*) in which the m^{th} function is *w[m]* diagonalization (see page 1128) yields the function *w[x][x]* shown below which cannot be primitive recursive. It is inevitable that the function shown must eventually grow faster than any primitive recursive function (at $x = 356$ its value is 63190, while at $x = 1464$ it is 1073844). But by reducing the results modulo 2 one gets a function that does not grow—and has seemingly quite random behavior—yet is presumably again not primitive recursive.

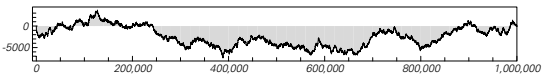


(Note that multiple arguments to a recursive function can be encoded as a single argument using functions like the β of page 1120—though the irregularity of such functions tends to make it difficult then to tell what is going on in the underlying recursive function.)

■ **Ulam sequences.** Slightly more complicated definitions in terms of numbers yield all sorts of sequences with very complicated forms. An example suggested by Stanislaw Ulam around 1960 (in a peculiar attempt to get a 1D analog of a 2D cellular automaton; see pages 877 and 928) starts with *{1, 2}*, then successively appends the smallest number that is the sum of two previous numbers in just one way, yielding

{1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, ...}

With this initial condition, the sequence is known to go on forever. At least up to $n = 10^6$ terms, it increases roughly like $13.5n$, but as shown below the fluctuations seem random.



The Sequence of Primes

■ **History of primes.** Whether the Babylonians had the notion of primes is not clear, but before 400 BC the Pythagoreans had introduced primes as numbers of objects that can be

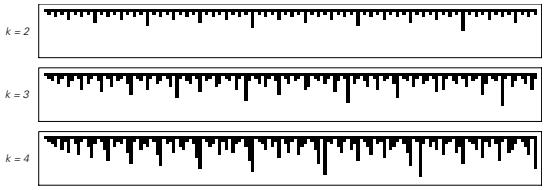
arranged only in a single line, and not in any other rectangular array. Around 300 BC Euclid discussed various properties of primes in his *Elements*, giving for example a proof that there are an infinity of primes. The sieve of Eratosthenes was described in 200 BC, apparently following ideas of Plato. Then starting in the early 1600s various methods for factoring were developed, and conjectures about formulas for primes were made. Pierre Fermat suggested $2^{2^n}+1$ as a source for primes and Marin Mersenne `2^Prime[n]-1` (see page 911). In 1752 Christian Goldbach showed that no ordinary polynomial could generate only primes, though as pointed out by Leonhard Euler n^2-n+41 does so for $n<40$. (With *If* or *Floor* included there are at least complicated cases known where polynomial-like formulas can be set up whose evaluation corresponds to explicit prime-generating procedures—see page 1162.) Starting around 1800 extensive work was done on analytical approximations to the distribution of primes (see below). There continued to be slow progress in finding specific large primes; $2^{31}-1$ was found prime around 1750 and $2^{127}-1$ in 1876. ($2^{2^5}+1$ was found composite in 1732, as have now all $2^{2^n}+1$ for $n \le 32$.) Then starting in the 1950s with the use of electronic computers many new large primes were found. The number of digits in the largest known prime has historically increased roughly exponentially with time over the past two decades, with a prime of over 4 million digits ($2^{13466917}-1$) now being known (see page 911).

■ **Page 132 · Finding primes.** The sieve of Eratosthenes shown in the picture is an appropriate procedure if one wants to find every prime, but testing whether an individual number is prime can be done much more efficiently, as in *PrimeQ[n]* in *Mathematica*, for example by using Fermat's so-called little theorem that $\text{Mod}[a^{p-1}, p] == 1$ whenever p is prime. The n^{th} prime *Prime[n]* can also be computed fairly efficiently using ideas from analytic number theory (see below).

■ **Decimation systems.** A somewhat similar system starts with a line of cells, then at each step removes every k^{th} cell that remains, as in the pictures below. The number of steps for which a cell at position n will survive can be computed as

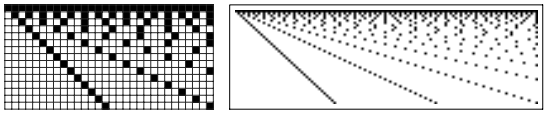
```
Module[{q = n + k - 1, s = 1},
  While[Mod[q, k] ≠ 0, q = Ceiling[(k - 1) q/k]; s++]; s]
```

If a cell is going to survive for s steps, then it turns out that this can be determined by looking at the last s digits in the base k representation of its position. For $k=2$, a cell survives for s steps if these digits are all 0 (so that s==*IntegerExponent[n, k]*). But for $k>2$, no such simple characterization appears to exist.



If the cells are arranged on a circle of size n, the question of which cell is removed last is the so-called Josephus problem. The solution is `Fold[Mod[#1 + k, #2, 1] &, 0, Range[n]]`, or `FromDigits[RotateLeft[IntegerDigits[n, 2]], 2]` for $k=2$.

■ **Page 132 · Divisors.** The picture below shows as black squares the divisors of each successive number (which correspond to the gray dots in the picture in the main text). Primes have divisors 1 and n only. (See also pages 902 and 747.)



■ **Page 133 · Results about primes.** *Prime[n]* is given approximately by $n\,\text{Log}[n]+n\,\text{Log}[\text{Log}[n]]$. (*Prime[$10^9$]* is 22,801,763,489 while the approximation gives 2.38×10^{10}.) A first approximation to *PrimePi[n]* is *n/Log[n]*. A somewhat better approximation is *LogIntegral[n]*, equal to *Integrate[1/Log[t], {t, 2, n}]*. This was found empirically by Carl Friedrich Gauss in 1792, based on looking at a table of primes. (*PrimePi[10^9]* is 50,847,534 while *LogIntegral[10^9]* is about 50,849,235.) A still better approximation is obtained by subtracting *Sum[LogIntegral[n^{r_i}], {i, -∞, ∞}]* where the r_i are the complex zeros of the Riemann zeta function *Zeta[s]*, discussed on page 918. According to the Riemann Hypothesis, the difference between *PrimePi[n]* and *LogIntegral[n]* is of order $\sqrt{n}\,\text{Log}[n]$. More refined analytical estimates of *PrimePi[n]* are good enough that they are used by *Mathematica* to compute *Prime[n]* for large n.

It is known that the ratio of the number of primes of the form $4k+1$ and $4k+3$ asymptotically approaches 1, but almost nothing has been proved about the fluctuations.

The gap between successive primes *Prime[n] - Prime[n - 1]* is thought to grow on average at most like $\text{Log}[\text{Prime}[n]]^2$. It is known that for sufficiently large n a gap of any size must exist. It is believed but not proved that there are an infinite number of "twin primes" with a gap of exactly 2.

■ **History of number theory.** Most areas of mathematics go from inception to maturity within at most a century. But in

number theory there are questions that were formulated more than 2000 years ago (such as whether any odd perfect numbers exist) that have still not been answered. Of the principles that have been established in number theory, a great many were first revealed by explicit experiments. From its inception in classical times, through its development in the 1600s to 1800s, number theory was largely separate from other fields of mathematics. But starting at the end of the 1800s, increasing connections were found to other areas of both continuous and discrete mathematics. And through these connections, sophisticated proofs of such results as Fermat's Last Theorem—open for 350 years—have been constructed. Long considered a rather esoteric branch of mathematics, number theory has in recent years grown in practical importance through its use in areas such as coding theory, cryptography and statistical mechanics. Properties of numbers and certain elementary aspects of number theory have also always played a central role in amateur and recreational mathematics. And as this chapter indicates, number theory can also be used to provide many examples of the basic phenomena discussed in this book.

■ **Page 134 · Tables of primes.** No explicit tables of primes appear to have survived from antiquity, but it seems likely that all primes up to somewhere between 5000 and 10000 were known. (In 348 BC, Plato mentioned divisors of 5040, and by 100 AD there is evidence that the fifth perfect number was known, requiring the knowledge that 8191 is prime.) In 1202 Leonardo Fibonacci explicitly gave as an example a list of primes up to 100. And by the mid-1600s there were printed tables of primes up to 100,000, containing as much data as in plots (c) and (d). In the 1700s and 1800s many tables of number factorizations were constructed; by the 1770s there was a table up to 2 million, and by the 1860s up to 100 million. A table of primes up to a trillion could now be generated fairly easily with current computer technology—though for most purposes computation of specific primes is more useful.

■ **Page 134 · Numbers of primes.** The fact that curve (c) must cross the axis was proved by John Littlewood in 1914, and it is known to have at least one crossing below 10^{317}. Somewhat related to the curves shown here is the function `MoebiusMu[n]`, equal to 0 if n has a repeated prime factor and otherwise `(-1)^Length[FactorInteger[n]]`. The quantity `FoldList[Plus, 0, Table[MoebiusMu[i], {i, n}]]` behaves very much like a random walk. The so-called Mertens Conjecture from 1897 stated that the magnitude of this quantity is less than $\sqrt{n}$. But this was disproved in 1983, although the necessary n is not known explicitly.

■ **Relative primes.** A single number is prime if it has no non-trivial factors. Two numbers are said to be relatively prime if they share no non-trivial factors. The pattern formed by numbers with this property is shown on page 613.

■ **Page 135 · Properties.** (a) The number of divisors of n is given by `DivisorSigma[0, n]`, equal to `Length[Divisors[n]]`. For large n this number is on average of order `Log[n] + 2 EulerGamma - 1`.

(b) *(Aliquot sums)* The quantity that is plotted is `DivisorSigma[1, n] - 2 n`, equal to `Apply[Plus, Divisors[n]] - 2 n`. This quantity was considered of great significance in antiquity, particularly by the Pythagoreans. Numbers were known as abundant, deficient or perfect depending on whether the quantity was positive, negative or zero. (See notes on perfect numbers below.) For large n, `DivisorSigma[1, n]` is known to grow at most like `Log[Log[n]] n Exp[EulerGamma]`, and on average like $\pi^2 n/6$ (see page 1093). As discovered by Srinivasa Ramanujan in 1918 its fluctuations (see below) can be obtained from the formula

```
1/6 π^2 n Sum[Apply[Plus, Cos[2 π n Select[
                Range[s], GCD[s, #] == 1 &]/s]]/s^2, {s, ∞}]
```

(c) Squares are taken to be of positive or negative integers, or zero. The number of ways of expressing an integer n as the sum of two such squares is `4 Apply[Plus, Im[i^Divisors[n]]]`. This is nonzero when all prime factors of n of the form $4k+3$ appear with even exponents. There is no known simple formula for the number of ways of expressing an integer as a sum of three squares, although part of the condition in the main text for integers to be expressible in this way was established by René Descartes in 1638 and the rest by Adrien Legendre in 1798. Note that the total number of integers less than n which can be expressed as a sum of three squares increases roughly like $5n/6$, with fluctuations related to `IntegerDigits[n, 4]`. It is known that the directions of all vectors `{x, y, z}` for which $x^2 + y^2 + z^2 == n$ are uniformly distributed in the limit of large n.

The total number of ways that integers less than n can be expressed as a sum of d squares is equal to the number of integer lattice points that lie inside a sphere of radius $\sqrt{n}$ in d-dimensional space. For $d = 2$, this approaches πn for large n, with an error of order n^c, where $1/4 < c \lesssim 0.315$.

(d) All numbers n can be expressed as the sum of four squares, in exactly `8 Apply[Plus, Select[Divisors[n], Mod[#, 4] ≠ 0 &]]` ways, as established by Carl Jacobi in 1829. Edward Waring stated in 1770 that any number can be expressed as a sum of at most 9 cubes and 19 fourth powers. Seven cubes appear to suffice for all but 17 numbers, the last of which is 455; four

cubes may suffice for all but 113936676 numbers, the last of which is 7373170279850. (See also page 1166.)

(e) Goldbach's Conjecture has been verified for all even numbers up to 4×10^{14}. In 1973 it was proved that any even number can be written as the sum of a prime and a number that has at most two prime factors, not necessarily distinct. The number of ways of writing an integer n as a sum of two primes can be calculated explicitly as *Length[Select[n - Table[Prime[i], {i, PrimePi[n]}], PrimeQ]]*. This quantity was conjectured by G. H. Hardy and John Littlewood in 1922 to be proportional to

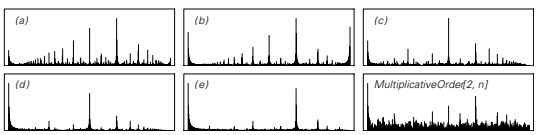
```
2 n Apply[Times, Map[(# - 1)/(# - 2) &,
        Map[First, Rest[FactorInteger[n]]]]]/Log[n]^2
```

It was proved in 1937 by Ivan Vinogradov that any large odd integer can be expressed as a sum of three primes.

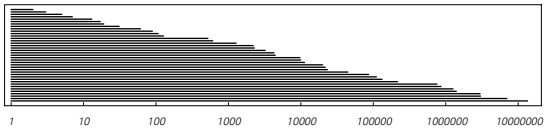
■ **Trapezoidal primes.** If one lays out n objects in an $a \times b$ rectangular array, then n is prime if either a or b must be 1. Following the Pythagorean idea of figurate numbers one can instead consider laying out objects in an array of b rows, containing successively a, $a-1$, … objects. It turns out all numbers except powers of 2 can be represented this way.

■ **Other integer functions.** *IntegerExponent[n, k]* gives nested behavior as for decimation systems on page 909, while *MultiplicativeOrder[k, n]* and *EulerPhi[n]* yield more complicated behavior, as shown on pages 257 and 1093.

■ **Spectra.** The pictures below show frequency spectra obtained from the sequences in the main text. Some regularity is evident, and in cases (a) and (b) it can be understood from trigonometric sum formulas of Ramanujan discussed above (see also pages 586 and 1081).

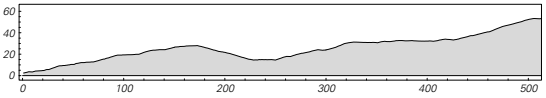


■ **Perfect numbers.** Perfect numbers with the property that *Apply[Plus, Divisors[n]] == 2 n* have been studied since at least the time of Pythagoras around 500 BC. The first few perfect numbers are *{6, 28, 496, 8128, 33550336}* (a total of 39 are currently known). It was shown by Euclid in 300 BC that $2^{n-1}(2^n - 1)$ is a perfect number whenever $2^n - 1$ is prime. Leonhard Euler then proved around 1780 that every even perfect number must have this form. The values of n for the known Mersenne primes $2^n - 1$ are shown below. These values can be found using the so-called Lucas-Lehmer test *Nest[Mod[#^2 - 2, 2^n - 1] &, 4, n - 2] == 0*, and in all cases n itself must be prime.



Whether any odd perfect numbers exist is probably the single oldest unsolved problem in mathematics. It is known that any odd perfect number must be greater than 10^{300}, must have a factor of at least 10^6, and must be less than 4^{4^s} if it has only s prime factors. Looking at curve (b) on page 135, however, it does not seem inconceivable that an odd perfect number could exist. For odd n up to 500 million the only values near 0 that appear in the curve are *{-6, -5, -4, -2, -1, 6, 18, 26, 30, 36}*, with, for example, the first 6 occurring at $n = 8925$ and last 18 occurring at $n = 159030135$. Various generalizations of perfect numbers have been considered, requiring for example *IntegerQ[DivisorSigma[1, n]/n]* (pluperfect) or *Abs[DivisorSigma[1, n] - 2 n] < r* (quasiperfect).

■ **Iterated aliquot sums.** Related to case (b) above is a system which repeats the replacement $n \to$ *Apply[Plus, Divisors[n]] - n* or equivalently $n \to$ *DivisorSigma[1, n] - n*. The fixed points of this procedure are the perfect numbers (see above). Other numbers usually evolve to perfect numbers, or to short repetitive sequences of numbers. But if one starts, for example, with the number 276, then the picture below shows the number of base 10 digits in the value obtained at each step.



After 500 steps, the value is the 53-digit number

39448887705043893375102470161238803295318090278129552

The question of whether such values can increase forever was considered by Eugène Catalan in 1887, and has remained unresolved since.

Mathematical Constants

■ **Page 137 · Digits of pi.** The digits of π shown here can be obtained in less than a second from *Mathematica* on a typical current computer using *N[π, 7000]*. Historically, the number of decimal digits of π that have been computed is roughly as follows: 2000 BC (Babylonians, Egyptians): 2 digits; 200 BC (Archimedes): 5 digits; 1430 AD: 14 digits; 1610: 35 digits; 1706: 100 digits; 1844: 200 digits; 1855: 500 digits; 1949 (ENIAC computer): 2037 digits; 1961: 100,000 digits (IBM 7090); 1973: 1 million; 1983: 16 million; 1989: 1 billion; 1997: 50 billion; 1999: 206 billion. In the first 200

billion digits, the frequencies of 0 through 9 differ from 20 billion by

```
{30841, -85289, 136978, 69393, -78309,
  -82947, -118485, -32406, 291044, -130820}
```

An early approximation to π was

```
4 Sum[(-1)^k/(2 k + 1), {k, 0, m}]
```

30 digits were obtained with

```
2 Apply[Times, 2/Rest[NestList[Sqrt[2 + #] &, 0, m]]]
```

An efficient way to compute π to n digits of precision is

```
(#[[2]]^2/#[[3]] &)[NestWhile[Apply[Function[{a, b, c, d},
      {(a + b)/2, Sqrt[a b], c - d (a - b)^2, 2 d}], #] &,
  {1, 1/Sqrt[N[2, n]], 1/4, 1/4}, #[[1]] ≠ #[[2]] &]]
```

This requires about *Log[2, n]* steps, or a total of roughly $n \, \text{Log}[n]^2$ operations (see page 1134).

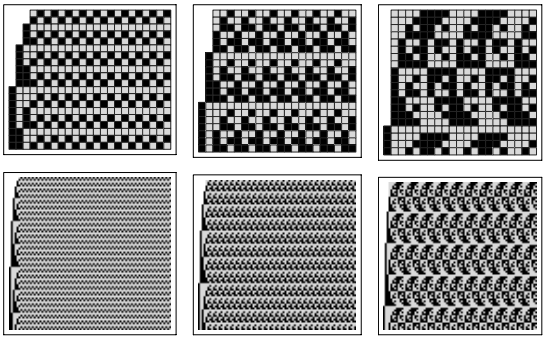
■ **Computing n^{th} digits directly.** Most methods for computing mathematical constants progressively generate each additional digit. But following work by Simon Plouffe and others in 1995 it became clear that it is sometimes possible to generate, at least with overwhelming probability, the n^{th} digit without explicitly finding previous ones. As an example, the n^{th} digit of *Log[2]* in base 2 is formally given by *Round[FractionalPart[2^n Sum[2^-k/k, {k, ∞}]]]*. And in practice the n^{th} digit can be found just by computing slightly over n terms of the sum, according to

```
Round[FractionalPart[
    Sum[FractionalPart[PowerMod[2, n - k, k]/k], {k, n}] +
      Sum[2^(n-k)/k, {k, n + 1, n + d}]]]
```

where several values of d can be tried to check that the result does not change. (Note that with finite-precision arithmetic, some exponentially small probability exists that truncation of numbers will lead to incorrect results.) The same basic approach as for *Log[2]* can be used to obtain base 16 digits in π from the following formula for π:

```
Sum[16^-k (4/(8 k + 1) - 2/(8 k + 4) -
      1/(8 k + 5) - 1/(8 k + 6)), {k, 0, ∞}]
```

A similar approach can also be used for many other constants that can be viewed as related to values of *PolyLog*.



■ **Page 139 · Rational numbers.** The pictures above show the base 2 digit sequences of numbers m/n for successive m.

The digits of $1/n$ in base b repeat with period

```
MultiplicativeOrder[b, FixedPoint[#/GCD[#, b] &, n]]
```

which is equal to *MultiplicativeOrder[b, n]* for prime n, and is at most $n - 1$. Each repeating block of digits typically seems quite random, and has properties such as all possible subblocks of digits up to a certain length appearing (see page 1084).

■ **Page 139 · Digit sequence properties.** Empirical evidence for the randomness of the digit sequences of $\sqrt{n}$, π, etc. has been accumulating since early computer experiments in the 1940s. The evidence is based on applying various standard statistical tests of randomness, and remains somewhat haphazard. (Already in 1888 John Venn had noted for example that the first 707 digits of π lead to an apparently typical 2D random walk.) (See page 1089.)

The fact that $\sqrt{2}$ is not a rational number was discovered by the Pythagoreans. Numbers that arise as solutions of polynomial equations are called algebraic; those that do not are called transcendental. e and π were proved to be transcendental in 1873 and 1882 respectively. It is known that *Exp[n]* and *Log[n]* for whole numbers n (except 0 and 1 respectively) are transcendental. It is also known for example that *Gamma[1/3]* and *BesselJ[0, n]* are transcendental. It is not known for example whether *EulerGamma* is even irrational.

A number is said to be "normal" in a particular base if every digit and every block of digits of any length occur with equal frequency. Note that the fact that a number is normal in one base does not imply anything about its normality in another base (unless the bases are related for example by both being powers of 2). Despite empirical evidence, no number expressed just in terms of standard mathematical functions has ever been rigorously proved to be normal. It has nevertheless been known since the work of Emile Borel in 1909 that numbers picked randomly on the basis of their value are almost always normal. And indeed with explicit constructions in terms of digits, it is quite straightforward to get numbers that are normal. An example of this is the number 0.1234567891011121314... obtained by concatenating the digits of successive integers in base 10 (see below). This number was discussed by David Champernowne in 1933, and is known to be transcendental. A few other results are also known. One based on gradual extension of work by Richard Stoneham from 1971 is that numbers of the form *Sum[1/(p^n b^(p^n)), {n, ∞}]* for prime $p > 2$ are normal in base b (for *GCD[b, p] == 1*), and are transcendental.

■ **Page 141 · Square roots.** A standard way to compute $\sqrt{n}$ is Newton's method (actually used already in 2000 BC by the Babylonians), in which one takes an estimate of the value x and then successively applies the rule $x \to 1/2\,(x + n/x)$. After t steps, this method yields a result accurate to about t^2 digits.

Another approach to computing square roots is based on the fact that the ratio of successive terms in for example the sequence $f[i] = 2\,f[i-1] + f[i-2]$ with $f[1] = f[2] = 1$ tends to $1 + \sqrt{2}$. This method yields about $2.5\,t$ base 2 digits after t steps.

The method of computing square roots shown in the main text is less efficient (it computes t digits in t steps), but illustrates more of the mechanisms involved. The basic idea is at every step t to maintain the relation $s^2 + 4\,r == 4^t\,n$, keeping r as small as possible so as to make $s \le 2^t\,\sqrt{n} < s + 4$. Note that the method works not only for integers, but for any rational number n for which $1 \le n < 4$.

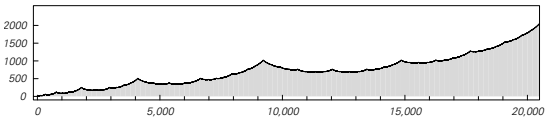
■ **Nested digit sequences.** The number obtained from the substitution system $\{1 \to \{1, 0\}, 0 \to \{0, 1\}\}$ is approximately 0.587545966 in base 10. It is certainly conceivable that a quantity such as Feigenbaum's constant (approximately 4.6692016091) could have a digit sequence with this kind of nested structure.

From the result on page 890, the number whose digits are obtained from $\{1 \to \{1, 0\}, 0 \to \{1\}\}$ is given by `Sum[2^(-Floor[n GoldenRatio]), {n, ∞}]`. This number is known to be transcendental. The n^{th} term in its continued fraction representation turns out to be `2^Fibonacci[n-2]`.

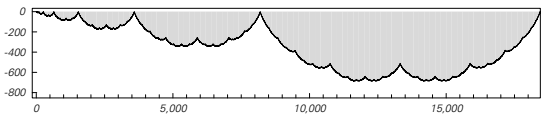
The fact that nested digit sequences do not correspond to algebraic numbers follows from work by Alfred van der Poorten and others in the early 1980s. The argument is based on showing that an algebraic function always exists for which the coefficients in its power series correspond to any given nested sequence when reduced modulo some p. (See page 1092.) But then there is a general result that if a particular sequence of power series coefficients can be obtained from an algebraic (but not rational) function modulo a particular p, then it can only be obtained from transcendental functions modulo any other p—or over the integers.

■ **Concatenation sequences.** One can consider forming sequences by concatenating digits of successive integers in base k, as in `Flatten[Table[IntegerDigits[i, k], {i, n}]]`. In the limit, such sequences contain with equal frequency all possible blocks of any given length, but as shown on page 597, they exhibit other obvious deviations from randomness. The picture below shows the $k = 2$ sequence chopped into length 256 blocks.

Applying `FoldList[Plus, 0, 2 list - 1]` to the whole sequence yields the pattern shown below.



The systematic increase is a consequence of the leading 1 in each concatenated sequence. Dropping this 1 yields the pattern below.



This is similar to picture (c) on page 131, and is a digit-by-digit version of

```
FoldList[Plus, 0,
  Table[Apply[Plus, 2 Rest[IntegerDigits[i, 2]] - 1], {i, n}]]
```

Note that although the picture above has a nested structure, the original concatenation sequences are not nested, and so cannot be generated by substitution systems. The element at position n in the first sequence discussed above can however be obtained in about `Log[n]` steps using

```
((IntegerDigits[#3 + Quotient[#1, #2], 2][[
        Mod[#1, #2] + 1]] &)[n - (# - 2) 2^(#-1) - 2, #,
    2^(#-1)] &)[NestWhile[# + 1 &, 0, (# - 1) 2^# + 1 < n &]]
```

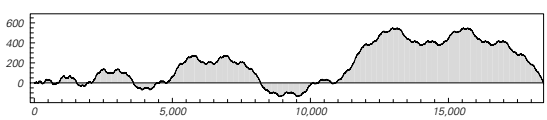
where the result of the `NestWhile` can be expressed as

```
Ceiling[1 + ProductLog[1/2 (n - 1) Log[2]]/Log[2]]
```

Following work by Maxim Rytin in the late 1990s about k^{n+1} digits of a concatenation sequence can be found fairly efficiently from

```
k/(k - 1)^2 -
  (k - 1) Sum[k^((k^s-1) (1+s-s k)/(k-1)) (1/((k - 1) (k^s - 1)^2) -
        k/((k - 1) (k^(s+1) - 1)^2) + 1/(k^(s+1) - 1)), {s, n}]
```

Concatenation sequences can also be generated by joining together digits from other representations of numbers; the picture below shows results for the Gray code representation from page 901.



■ **Specially constructed transcendental numbers.** Numbers known to be transcendental include ones whose digit sequences contain 1's only at positions $n!$, 2^n or *Fibonacci[n]*. Concatenation sequences, as well as generalizations formed by concatenating values of polynomials at successive integer points, are also known to yield numbers that are transcendental.

■ **Runs of digits.** One can consider any base 2 digit sequence as consisting of successive runs of 0's and 1's, constructed from the list of run lengths by

```
Fold[Join[#1, Table[1 - Last[#1], {#2}]] &, {0}, list]
```

This representation is related to so-called surreal numbers (though with the first few digits different). The number with run lengths corresponding to successive integers (so that the n^{th} digit is *Mod[Floor[1/2 + Sqrt[2 n]], 2]*) turns out to be $(1 - 2^{1/4}\ \text{EllipticTheta}[2, 0, 1/2] + \text{EllipticTheta}[3, 0, 1/2])/2$, and appears at least not to be algebraic.

■ **Leading digits.** Even though in individual numbers generated by simple mathematical procedures all possible digits often appear to occur with equal frequency, leading digits in sequences of numbers typically do not. Instead it is common for a leading digit s in base b to occur with frequency *Log[b, (s + 1)/s]* (so that in base 10 1's occur 30% of the time and 9's 4.5%). This will happen whenever *FractionalPart[Log[b, a[n]]]* is uniformly distributed, which, as discussed on page 903, is known to be true for sequences such as r^n (with *Log[b, r]* irrational), n^n, $n!$, *Fibonacci[n]*, but not $r\,n$, *Prime[n]* or *Log[n]*. A logarithmic law for leading digits is also found in many practical numerical tables, as noted by Simon Newcomb in 1881 and Frank Benford in 1938.

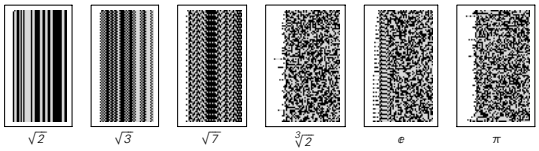
■ **Page 143 · Continued fractions.** The first n terms in the continued fraction representation for a number x can be found from the built-in *Mathematica* function *ContinuedFraction*, or from

```
Floor[NestList[1/Mod[#, 1] &, x, n - 1]]
```

A rational approximation to the number x can be reconstructed from the continued fraction using *FromContinuedFraction* or by

```
Fold[1/#1 + #2 &, Last[list], Rest[Reverse[list]]]
```

The pictures below show the digit sequences of successive iterates obtained from *NestList[1/Mod[#, 1] &, x, n]* for several numbers x.



Unlike ordinary digits, the individual terms in a continued fraction can be of any size. In the continued fraction for a randomly chosen number, the probability to find a term of size s is *Log[2, (1 + 1/s)/(1 + 1/(s + 1))]*, so that the probability of getting a 1 is about 41.50%, and the probability of getting a large term falls off like $1/s^2$. If one looks at many terms, then their geometric mean is finite, and approaches Khinchin's constant $\text{Khinchin} \approx 2.68545$.

In the first 1000 terms of the continued fraction for π, there are 412 1's, and the geometric mean is about 2.6656. The largest individual term is the 432th one, which is equal to 20,776. In the first million terms, there are 414,526 1's, the geometric mean is 2.68447, and the largest term is the 453,294th one, which is 12,996,958.

Note that although the usual continued fraction for π looks quite random, modified forms such as

```
4/(Fold[#2/#1 + 2 &, 2, Reverse[Range[1, n, 2]^2]] - 1)
```

can be very regular.

The continued fractions for *Exp[2/k]* and *Tan[k/2]* have simple forms (as discussed by Leonhard Euler in the mid-1700s); other rational powers of e and tangents do not appear to. The sequence of odd numbers gives the continued fraction for *Coth[1]*; the sequence of even numbers for *BesselI[0, 1]/BesselI[1, 1]*. In general, continued fractions whose n^{th} term is $a\,n + b$ correspond to numbers given by *BesselI[b/a, 2/a]/BesselI[b/a + 1, 2/a]*. Numbers whose continued fraction terms are polynomials in n can presumably also be represented in terms of suitably generalized hypergeometric functions. (All so-called Hurwitz numbers have continued fractions that consist of interleaved polynomial sequences—a property left unchanged by $x \to (a\,x + b)/(c\,x + d)$.)

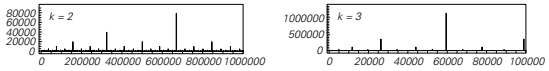
As discovered by Jeffrey Shallit in 1979, numbers of the form *Sum[1/k^i, {i, 0, ∞}]* that have nonzero digits in base k only at positions 2^i turn out to have continued fractions with terms of limited size, and with a nested structure that can be found using a substitution system according to

```
{0, k - 1, k + 2, k, k, k - 2, k, k + 2, k - 2, k}[[
  Nest[Flatten[{{1, 2}, {3, 4}, {5, 6}, {7, 8}, {5, 6}, {3, 4},
      {9, 10}, {7, 8}, {9, 10}, {3, 4}}[[#]]] &, 1, n]]
```

The continued fractions for square roots are always periodic; for higher roots they never appear to show any significant regularities. The first million terms in the continued fraction for $2^{1/3}$ contain 414,983 1's, have geometric mean 2.68505, and have largest term 4,156,269 at position 484,709. Terms of any size presumably in the end always occur in continued fractions for higher roots, though this is not known for certain. Fairly large terms are sometimes seen quite early: in $5^{1/3}$ term 19 is 3052, while in *Root[10 + 8 # - #^3 &, 1]* term 34

is 1,501,790. The presence of a large term indicates a close approximation to a rational number. In a few known cases simple formulas yield numbers that are close but not equal to integers. An example discovered by Srinivasa Ramanujan around 1913 is *Exp[π √163]*, which is an integer to one part in 10^{30}, and has second continued fraction term 1,333,462,407,511. (This particular example can be understood from the fact that as *d* increases *Exp[π √d]* becomes extremely close to *-1728 KleinInvariantJ[(1 + √-d)/2]*, which turns out to be an integer whenever there is unique factorization of numbers of the form $a + b\sqrt{-d}$ —and *d = 163* is the largest of the 9 cases for which this is so.) Other less spectacular examples include *Exp[π] - π* and *163/Log[163]*.

Numbers with digits given by concatenation sequences in any base *k* (see note above) seem to have unusual continued fractions, in which most terms are fairly small, but some are extremely large. Thus with *k = 2*, term 30 is 4,534,532, term 64 is 4,682,854,730,443,938, term 152 is about 2×10^{34} and term 669,468 is about 2×10^{78902}. (For the *k = 10* case of the original Champernowne number, even term 18 is already about 5×10^{165}.) The plots below of the numbers of digits in successive terms turn out to have patterns of peaks that show some signs of nesting.



In analogy to digits in a concatenation sequence the terms in the sequence

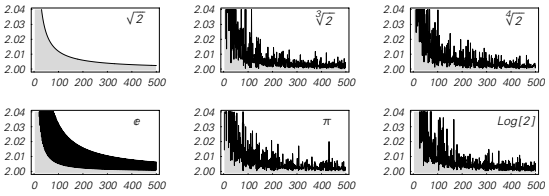
```
Flatten[Table[Rest[ContinuedFraction[a/b]],
                                {b, 2, n}, {a, b - 1}]]
```

are known to occur with the same frequencies as they would in the continued fraction representation for a randomly chosen number.

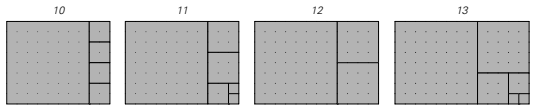
The pictures below show as a function of *n* the quantity

```
With[{r = FromContinuedFraction[ContinuedFraction[x, n]]},
                        -Log[Denominator[r], Abs[x - r]]]
```

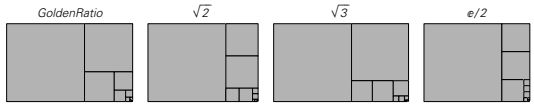
which gives a measure of the closeness of successive rational approximations to *x*. For any irrational number this quantity cannot be less than 2, while for algebraic irrationals Klaus Roth showed in 1955 that it can only have finitely many peaks that reach above any specified level.



▪ **History.** Euclid's algorithm states that starting from integers *{a, b}* iterating *{a_, b_} :→ If[a > b, {a - b, b}, {a, b - a}]* eventually leads to *{GCD[a, b], 0}*. (See page 1093.) The pictures below show how this works. The numbers of successively smaller squares (corresponding to the numbers of steps in the algorithm) turn out to be exactly *ContinuedFraction[a/b]*.



It was discovered in antiquity that Euclid's algorithm starting with *{x, 1}* terminates only when *x* is rational. In all cases, however, the relationship with continued fractions remains, as below.



Infinite continued fractions appear to have first been explicitly written down in the mid-1500s, and to have become popular in many problems in number theory by the 1700s. Leonhard Euler studied many continued fractions, while Joseph Lagrange seems to have thought that it might be possible to recognize any algebraic number from its continued fraction. The periodicity of continued fractions for quadratic irrationals was proved by Evariste Galois in 1828. From the late 1800s interest in continued fractions as such waned; it finally increased again in the 1980s in connection with problems in dynamical systems theory.

▪ **Egyptian fractions.** Following the ancient Egyptian number system, rational numbers can be represented by sums of reciprocals, as in *3/7 == 1/3 + 1/11 + 1/231*. With suitable distinct integers *a[n]* one can represent any number by *Sum[1/a[n], {n, ∞}]*. The representation is not unique; *a[n] = 2^n*, *n(n + 1)* and *(n + 1)!/n* all yield *1*. Simple choices for *a[n]* yield many standard transcendental numbers: *n!*: *e - 1*; *n!^2*: *BesselI[0, 2] - 1*; *n 2^n*: *Log[2]*; *n^2*: *π^2/6*; *(2 n - 1)(2 n - 3)*: *π √3 /9*; *3 - 16 n + 16 n^2*: *π/8*; *n n!*: *ExpIntegralEi[1] - EulerGamma*. (See also page 902.)

▪ **Nested radicals.** Given a list of integers acting like digits one can consider representing numbers in the form *Fold[Sqrt[#1 + #2] &, 0, Reverse[list]]*. A sequence of identical digits *d* then corresponds to the number *(1 + Sqrt[4 d + 1])/2*. (Note that *Nest[Sqrt[# + 2] &, 0, n] == 2 Cos[π/2^(n+1)]*.) Repeats of a digit block *b* give numbers that solve *Fold[#1^2 - #2 &, x, b] == x*. It appears that digits 0, 1, 2 are

sufficient to represent uniquely all numbers between 1 and 2. For any number x the first n digits are given by

```
Ceiling[NestList[(2 - Mod[-#, 1])^2 &, x^2, n - 1] - 2]
```

Even rational numbers such as $3/2$ do not yield simple digit sequences. For random x, digits 0, 1, 2 appear to occur with limiting frequencies $\sqrt{2+d} - \sqrt{1+d}$.

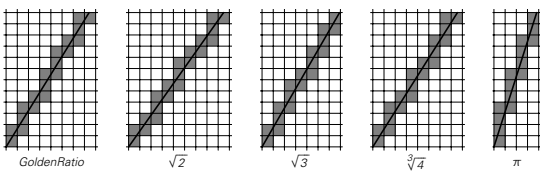
■ **Digital slope representation.** One can approximate a line of any slope h as in the picture below by a sequence of segments on a square grid (such as a digital display device). The vertical distance moved at the n^{th} horizontal position is $\text{Floor}[n\,h] - \text{Floor}[(n-1)\,h]$, and the sequence obtained from this (which contains only terms $\text{Floor}[h]$ and $\text{Floor}[h]+1$) provides a unique representation for h. As discussed on page 903 this sequence can be generated by applying substitution rules derived from the continued fraction form of h. If h is rational, the sequence is repetitive, while if h is a quadratic irrational, it is nested. Given a sequence of length n, an approximation to h can be reconstructed using

```
Max[MapIndexed[#1/First[#2] &,
    FoldList[Plus, First[list], Rest[list]]]]
```

The fractional part of the result obtained is always an element of the Farey sequence

```
Union[Flatten[Table[a/b, {b, n}, {a, 0, b}]]]
```

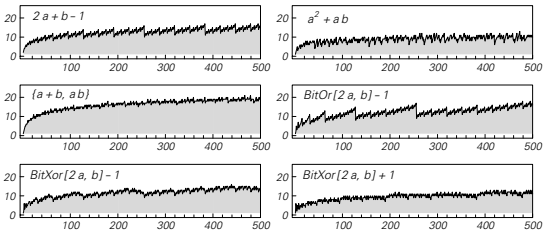
(See also pages 892, 932 and 1084.)



■ **Representations for integers.** See page 560.

■ **Operator representations.** Instead of repeatedly applying an operation to a sequence of digits one can consider forming integers (or other numbers) by performing trees of operations on a single constant. Thus, for example, any integer m can be obtained by a tree of $m-1$ additions of 1's such as $(1+(1+1))+1$. Another operator that can be used to generate any integer is $a \circ b = 2a+b-1$. In this case 6 is $(1\circ(1\circ 1))\circ 1$, and an integer m can be obtained by $\text{Tr}[1 + \text{IntegerDigits}[m, 2]] - 2$ or at most $\text{Log}[2, m]$ applications of $\circ$. The operator $k\,a+b-k+1$ can be used for any k. It also turns out that $\text{BitXor}[2\,a, b]+1$ works, though in this case even for 2 the smallest representation is $(1\circ 1)\circ(1\circ((1\circ 1)\circ 1))$. (For $\text{BitOr}[2\,a, b]-1$ the number of applications needed is $\text{With}[\{i = \text{IntegerDigits}[m, 2]\}, \text{Tr}[i + 1] + i[[2]]\,(1 + i[[3]]) - 1]$.)

The pictures below show the smallest number of operator applications required for successive integers. With the pair of operators $a+b$ and $a \times b$ (a case considered in recreational mathematics for n-ary operators) numbers of the form 3^s have particularly small representations. Note that in all cases the size of the smallest representation must at some level increase like $\text{Log}[m]$ (compare pages 1067 and 1070), but there may be some "algorithmically simple" integers that have shorter representations.



■ **Number classification.** One can imagine classifying real numbers in terms of what kinds of operations are needed to obtain them from integers. Rational numbers require only division (or solving linear equations), while algebraic numbers require solving polynomial equations. Rather little is known about numbers that require solving transcendental equations—and indeed it can even be undecidable (see page 1138) whether two equations can yield the same number. Starting with integers and then applying arithmetic operations and fractional powers one can readily reproduce all algebraic numbers up to degree 4, but not beyond. The sets of numbers that can be obtained by applying elementary functions like Exp, Log and Sin seem in various ways to be disjoint from algebraic numbers. But if one applies multivariate elliptic or hypergeometric functions it was established in the late 1800s and early 1900s that one can in principle reach any algebraic number. One can also ask what numbers can be generated by integrals (or by solving differential equations). For rational functions $f[x]$, $\text{Integrate}[f[x], \{x, 0, 1\}]$ must always be a linear function of Log and ArcTan applied to algebraic numbers ($f[x] = 1/(1+x^2)$ for example yields $\pi/4$). Multiple integrals of rational functions can be more complicated, as in

```
Integrate[1/(1 + x^2 + y^2), {x, 0, 1}, {y, 0, 1}] ==
  HypergeometricPFQ[{1/2, 1, 1}, {3/2, 3/2}, 1/9]/6 +
    1/2 π ArcSinh[1] - Catalan
```

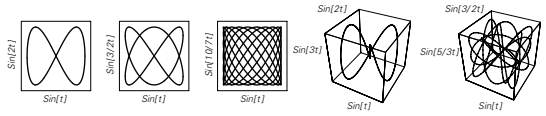
and presumably often cannot be expressed at all in terms of standard mathematical functions. Integrals of rational functions over regions defined by polynomial inequalities have recently been discussed under the name "periods". Many numbers associated with Zeta and Gamma can readily be generated, though apparently for example e and EulerGamma cannot. One can also consider numbers obtained from infinite sums (or by solving recurrence

equations). If *f[n]* is a rational function, *Sum[f[n], {n, ∞}]* must just be a linear combination of *PolyGamma* functions, but again the multivariate case can be much more complicated.

Mathematical Functions

▪ **Page 145 · Mathematical functions.** (See page 1091.) *BesselJ[0, x]* goes like $Sin[x]/\sqrt{x}$ for large *x* while *AiryAi[-x]* goes like $Sin[x^{3/2}]/x^{1/4}$. Other standard mathematical functions that oscillate at large *x* include *JacobiSN* and *MathieuC*. Most hypergeometric-type functions either increase or decrease exponentially for large arguments, though in the directions of Stokes lines in the complex plane they can oscillate sinusoidally. (For *AiryAi[x]* the Stokes lines are in directions *(-1)^({1, 2, 3}/3)*.)

▪ **Lissajous figures.** Plotting multiple sine functions each on different coordinate axes yields so-called Lissajous or Bowditch figures, as illustrated below. If the coefficients inside all the sine functions are rational, then going from *t = 0* to *t = 2 π Apply[LCM, Map[Denominator, list]]* yields a closed curve. Irrational ratios of coefficients lead to curves that never close and eventually fill space uniformly.

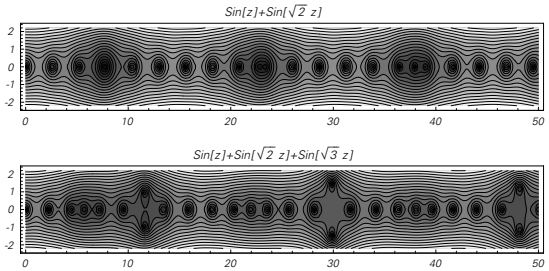


▪ **Page 146 · Two sine functions.** *Sin[a x] + Sin[b x]* can be rewritten as *2 Sin[1/2 (a + b) x] Cos[1/2 (a - b) x]* (using *TrigFactor*), implying that the function has two families of equally spaced zeros: $2\pi n/(a+b)$ and $2\pi (n+1/2)/(b-a)$.

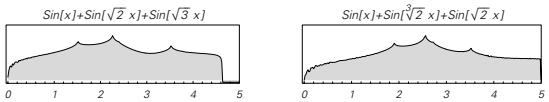
▪ **Differential equations.** The function $Sin[x] + Sin[\sqrt{2}\, x]$ can be obtained as the solution of the differential equation *y''[x] + 2 y[x] - Sin[x] == 0* with the initial conditions *y[0] == 0, y'[0] == 2*.

▪ **Musical chords.** In a so-called equal temperament scale the 12 standard musical notes that make up an octave have a progression of frequencies $2^{n/12}$. Most schemes for musical tuning use rational approximations to these numbers. Until the past century, and since at least the 1300s, diminished fifth or tritone chords that consist of two notes (such as C and G♭) with frequency ratio $\sqrt{2}$ have generally been avoided as sounding discordant. (See also page 1079.)

▪ **Page 146 · Three sine functions.** All zeros of the function *Sin[a x] + Sin[b x]* lie on the real axis. But for *Sin[a x] + Sin[b x] + Sin[c x]*, there are usually zeros off the real axis (even say for *a = 1, b = 3/2, c = 5/3*), as shown in the pictures below.



If *a*, *b* and *c* are rational, *Sin[a x] + Sin[b x] + Sin[c x]* is periodic with period *2 π/GCD[a, b, c]*, and there are a limited number of different spacings between zeros. But in a case like $Sin[x] + Sin[\sqrt{2}\, x] + Sin[\sqrt{3}\, x]$ there is a continuous distribution of spacings between zeros, as shown on a logarithmic scale below. (For $0 < x < 10^6$ there are a total of 448,494 zeros, with maximum spacing $\simeq 4.6$ and minimum spacing $\simeq 0.013$.)

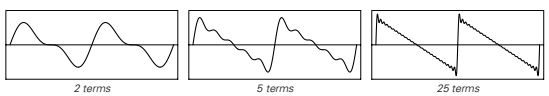


▪ **Page 147 · Substitution systems.** *Cos[a x] - Cos[b x]* has two families of zeros: $2\pi n/(a+b)$ and $2\pi n/(b-a)$. Assuming $b > a > 0$, the number of zeros from the second family which appear between the n^{th} and $n+1^{\text{th}}$ zero from the first family is

(Floor[(n + 1) #] - Floor[n #] &)[(b - a)/(a + b)]

and as discussed on page 903 this sequence can be obtained by applying a sequence of substitution rules. For *Sin[a x] + Sin[b x]* a more complicated sequence of substitution rules yields the analogous sequence in which *-1/2* is inserted in each *Floor*.

▪ **Many sine functions.** Adding many sine functions yields a so-called Fourier series (see page 1074). The pictures below show *Sum[Sin[n x]/n, {n, k}]* for various numbers of terms *k*. Apart from a glitch that gets narrower with increasing *k* (the so-called Gibbs phenomenon), the result has a simple triangular form. Other so-called Fourier series in which the coefficient of *Sin[m x]* is a smooth function of *m* for all integer *m* yield similarly simple results.

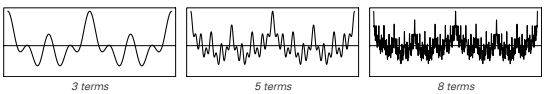


The pictures below show *Sum[Sin[n^2 x]/n^2, {n, k}]*, where in effect all coefficients of *Sin[m x]* other than those where *m* is

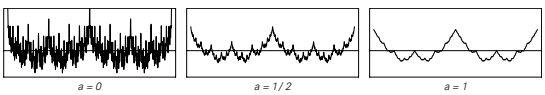
a perfect square are set to zero. The result is a much more complicated curve. Note that for x of the form $p\,\pi/q$, the $k = \infty$ sum is just

```
(π/(2 q))^2 Sum[Sin[n^2 p π/q]/Sin[n π/(2 q)]^2, {n, q - 1}]
```

The pictures below show `Sum[Cos[2^n x], {n, k}]` (as studied by Karl Weierstrass in 1872). The curves obtained in this case show a definite nested structure, in which the value at a point x is essentially determined directly from the base 2 digit sequence of x. (See also page 1080.)



The curves below are approximations to `Sum[Cos[2^n x]/2^(a n), {n, ∞}]`. They can be thought of as having dimensions $2 - a$ and smoothed power spectra $\omega^{-(1+2a)}$.



▪ **FM synthesis.** More complicated curves can be obtained for example using FM synthesis, as discussed on page 1079.

▪ **Page 148 · Zeta function.** For real s the Riemann zeta function `Zeta[s]` is given by `Sum[1/n^s, {n, ∞}]` or `Product[1/(1 - Prime[n]^s), {n, ∞}]`. The zeta function as analytically continued for complex s was studied by Bernhard Riemann in 1859, who showed that `PrimePi[n]` could be approximated (see page 909) up to order $\sqrt{n}$ by `LogIntegral[n] - Sum[LogIntegral[n^r[i]], {i, -∞, ∞}]`, where the `r[i]` are the complex zeros of `Zeta[s]`. The Riemann Hypothesis then states that all `r[i]` satisfy `Re[r[i]] == 1/2`, which implies a certain randomness in the distribution of prime numbers, and a bound of order $\sqrt{n}\,\mathrm{Log}[n]$ on `PrimePi[n] - LogIntegral[n]`. The Riemann Hypothesis is also equivalent to the statement that a bound of order $\sqrt{n}\,\mathrm{Log}[n]^2$ exists on `Abs[Log[Apply[LCM, Range[n]]] - n]`.

The picture in the main text shows `RiemannSiegelZ[t]`, defined as `Zeta[1/2 + i t] Exp[i RiemannSiegelTheta[t]]`, where

```
RiemannSiegelTheta[t_] =
  Arg[Gamma[1/4 + i t/2]] - 1/2 t Log[π]
```

The first term in an approximation to `RiemannSiegelZ[t]` is `2 Cos[RiemannSiegelTheta[t]]`; to get results to a given precision requires summing a number of terms that increases like $\sqrt{t}$, making routine computation possible up to $t \sim 10^{10}$.

It is known that:

- The average spacing between zeros decreases like `1/Log[t]`.
- The amplitude of wiggles grows with t, but more slowly than $t^{0.16}$.
- At least the first 10 billion zeros have `Re[s] == 1/2`.

The statistical distribution of zeros was studied by Andrew Odlyzko and others starting in the late 1970s (following ideas of David Hilbert and George Pólya in the early 1900s), and it was found that to a good approximation, the spacings between zeros are distributed like the spacings between eigenvalues of random unitary matrices (see page 977).

In 1972 Sergei Voronin showed that `Zeta[z + (3/4 + i t)]` has a certain universality in that there always in principle exists some t (presumably in practice usually astronomically large) for which it can reproduce to any specified precision over say the region `Abs[z] < 1/4` any analytic function without zeros.

Iterated Maps and the Chaos Phenomenon

▪ **History of iterated maps.** Newton's method from the late 1600s for finding roots of polynomials (already used in specific cases in antiquity) can be thought of as a smooth iterated map (see page 920) in which a rational function is repeatedly applied (see page 1101). Questions of convergence led in the late 1800s and early 1900s to interest in iteration theory, particularly for rational functions in the complex plane (see page 933). There were occasional comments about complicated behavior (notably by Arthur Cayley in 1879) but no real investigation seems to have been made. In the 1890s Henri Poincaré studied so-called return maps giving for example positions of objects on successive orbits. Starting in the 1930s iterated maps were sometimes considered as possible models in fields like population biology and business cycle theory—usually arising as discrete annualized versions of continuous equations like the Verhulst logistic differential equation from the mid-1800s. In most cases the most that was noted was simple oscillatory behavior, although for example in 1954 William Ricker iterated empirical reproduction curves for fish, and saw more complex behavior—though made little comment on it. In the 1950s Paul Stein and Stanislaw Ulam did an extensive computer study of various iterated maps of nonlinear functions. They concentrated on questions of convergence, but nevertheless noted complicated behavior. (Already in the

late 1940s John von Neumann had suggested using $x \to 4x(1-x)$ as a random number generator, commenting on its extraction of initial condition digits, as mentioned on page 921.) Some detailed analytical studies of logistic maps of the form $x \to ax(1-x)$ were done in the late 1950s and early 1960s—and in the mid-1970s iterated maps became popular, with much analysis and computer experimentation on them being done. But typically studies have concentrated on repetition, nesting and sensitive dependence on initial conditions—not on more general issues of complexity.

In connection with his study of continued fractions Carl Friedrich Gauss noted in 1799 complexity in the behavior of the iterated map $x \to$ *FractionalPart[1/x]*. Beginning in the late 1800s there was number theoretical investigation of the sequence *FractionalPart[a^n x]* associated with the map $x \to$ *FractionalPart[a x]* (see page 903), notably by G. H. Hardy and John Littlewood in 1914. Various features of randomness such as uniform distribution were established, and connections to smooth iterated maps emerged after the development of symbolic dynamics in the late 1930s.

■ **History of chaos theory.** See page 971.

■ **Page 150 · Exact iterates.** For any integer a the n^{th} iterate of $x \to$ *FractionalPart[a x]* can be written as *FractionalPart[a^n x]*, or equivalently *1/2 - ArcTan[Cot[a^n π x]]/π*. In the specific case $a = 2$ the iterates of *If[x < 1/2, a x, a(1-x)]* have the form *ArcCos[Cos[2^n π x]]/π*. (See pages 903 and 1098.)

■ **Page 151 · Problems with computer experiments.** The defining characteristic of a system that exhibits chaos is that on successive steps the system samples digits which lie further and further to the right in its initial condition. But in a practical computer, only a limited number of digits can ever be stored. In *Mathematica*, one can choose how many digits to store (and in the pictures shown in the main text, enough digits were used to avoid the problems discussed in this note). But a low-level language such as FORTRAN, C or Java always stores a fixed number of digits, typically around 53, in its standard double-precision floating-point representation of numbers.

So what happens when a system one is simulating tries to sample digits in its initial conditions beyond the ones that are stored? The answer depends on the way that arithmetic is handled in the computer system one uses.

When doing high-precision arithmetic, *Mathematica* follows the principle that it should only ever give digits that are known to be correct on the basis of the input that was provided. This means that in simulating chaotic systems, the numbers produced will typically have progressively fewer digits: later digits cannot be known to be correct without more precise knowledge of this initial condition. (An example is *NestList[Mod[2#, 1] &, N[π/4, 40], 200]*; *Map[Precision, list]* gives the number of significant digits of each element in the list.)

But most current languages and hardware systems follow a rather different approach. (For low-precision machine arithmetic, *Mathematica* is also forced to follow this approach.) What they do is to give a fixed number of digits as the result of every computation, whether or not all those digits are known to be correct. It is then the task of numerical analysis to establish that in a particular computation, the final results obtained are not unduly affected by digits that are not known to be correct. And in practice, for many kinds of computations, this is to a large extent the case. But whenever chaos is involved, it is inevitably not.

As an example, consider the iterated map $x \to$ *Mod[2x, 1]* discussed in the main text. At each step, this map shifts all the base 2 digits in x one position to the left. But if the computer gives a fixed number of digits at each step, then additional digits must be filled in on the right. On most computers, these additional digits are always 0. And so after some number of steps, all the digits in x are 0, and thus the value of x is simply 0.

But it turns out that a typical pocket calculator gives a different result. For pocket calculators effectively represent numbers in base 10 (actually so-called binary-coded decimal) not base 2, and fill in unknown digits with 0 in base 10. (Base 10 is used so that multiplying for example 1/3 by 3 gives exactly 1 rather than the more confusing result 0.9999... obtained with base 2.)

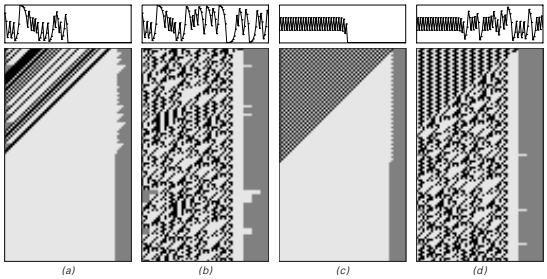
Pictures (a) and (c) below show simulations of the shift map on a typical computer, while pictures (b) and (d) show corresponding simulations on a pocket calculator. (Starting with initial condition x the digit sequence at step n is essentially

IntegerDigits[Mod[2^n Floor[2^{53} x], 2^{53}], 2, 53]

on the computer, and

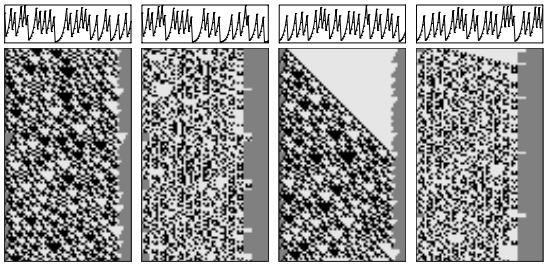
Flatten[IntegerDigits[IntegerDigits[
Mod[2^n Floor[10^{12} x], 10^{12}], 10, 12], 2, 4]]

on the calculator. In both cases the limited number of digits implies behavior that ultimately repeats—but only long after the other effects we discuss have occurred.)



For the first several steps, the results as shown at the top of each corresponding picture agree. But as soon as the effect of sampling beyond the digits explicitly stored in the initial condition becomes important, the results are completely different. The computer gives simply 0, but the pocket calculator yields apparently random sequences—which turn out to be analogous to those discussed on page 319.

Other chaotic systems have a similar sensitivity to the details of computer arithmetic. But the simple behavior of the shift map turns out to be rather rare: in most cases—such as the multiplication by 3/2 shown in the pictures below—apparent randomness is produced, even on a typical computer.



It is important to realize however that this randomness has little to do with the details of the initial conditions. Instead, just as in other examples in this book, the randomness arises from an intrinsic process that occurs even with the simple repetitive initial condition shown in pictures (c) and (d) above.

Computer simulations of chaotic systems have been done since the 1950s. And it has often been observed that the sequences generated in these simulations look quite random. But as we now see, such randomness cannot in fact be a consequence of the chaos phenomenon and of sensitive dependence on initial conditions.

Nevertheless, confusingly enough, even though it does not come from sensitive dependence on initial conditions, such randomness is what makes the overall properties of simulations typically follow the idealized mathematical predictions of chaos theory. The point is that the presence of randomness makes the system behave on different steps as if it were evolving from slightly different initial conditions. But statistical averages over different initial conditions typically yield essentially the results one would get by evolution from a single initial condition containing an infinite number of randomly chosen digits.

▪ **Page 152 · Mathematical perspectives.** Mathematicians may be confused by my discussion of complexity in iterated maps. The first point to make is that the issues I am studying are rather different from the ones that are traditionally studied in the mathematics of these systems. The next point is that I have specifically chosen not to make the idealizations about numbers and operations on numbers that are usually made in mathematics.

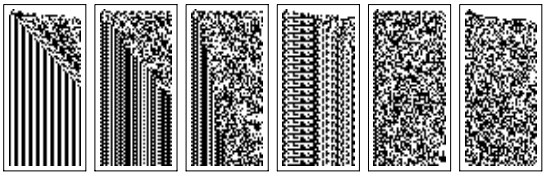
In particular, it is usually assumed that performing some standard mathematical operation, such as taking a square root, cannot have a significant effect on the system one is studying. But in trying to track down the origins of complex behavior, the effects of such operations can be significant. Indeed, as we saw on page 141, taking square roots can for example generate seemingly random digit sequences.

Many mathematicians may object that digit sequences are just too fragile an entity to be worth studying. They may argue that it is only robust and invariant concepts that are useful. But robustness with respect to mathematical operations is a different issue from robustness with respect to computational operations. Indeed, we will see later in this book that large classes of digit sequences can be considered equivalent with respect to computational operations, but these classes are quite different ones from those that are considered equivalent with respect to mathematical operations.

▪ **Information content of initial conditions.** Common sense suggests that it is a quite different thing to specify a simple initial condition containing, say, a single black cell on a white background, than to specify an initial condition containing an infinite sequence of randomly chosen cells. But in traditional mathematics no distinction is usually made between these kinds of specifications. And as a result, mathematicians may find it difficult to understand my distinction between randomness generated intrinsically by the evolution of a system and randomness from initial conditions (see page 299). The distinction may seem more obvious if one considers, for example, sequential substitution systems or cyclic tag systems. For such systems cannot meaningfully be given infinite random initial conditions, yet they can still perfectly well generate highly random behavior. (Their initial conditions correspond in a sense to integers rather than real numbers.)

▪ **Smooth iterated maps.** In the main text, all the functions used as mappings consist of linear pieces, usually joined together discontinuously. But the same basic phenomena seen with such mappings also occur when smooth functions are used. A particularly well-studied example (see page 918) is the so-called logistic map $x \to a\,x\,(1-x)$. The base 2 digit

sequences obtained with this map starting from $x = 1/8$ are shown below for various values of a. The quadratic nature of the map typically causes the total number of digits to double at each step. But at least for small a, progressively more digits on the left show purely repetitive behavior. As a increases, the repetition period goes through a series of doublings. The detailed behavior is different for every value of a, but whenever the repetition period is 2^j, it turns out that with any initial condition the leftmost digit always eventually follows a sequence that consists of repetitions of step j in the evolution of the substitution system $\{1 \to \{1, 0\}, 0 \to \{1, 1\}\}$ starting either from $\{0\}$ or $\{1\}$. As a approaches 3.569946, the period doublings get closer and closer together, and eventually a point is reached at which the sequence of leftmost digits is no longer repetitive but instead corresponds to the nested pattern formed after an infinite number of steps in the evolution of the substitution system. (An important result discovered by Mitchell Feigenbaum in 1975 is that this basic setup is universal to all smooth maps whose functions have a single hump.) When a is increased further, there is usually no longer repetitive or nested behavior. And although there are typically some constraints, the behavior obtained tends to depend on the details of the digit sequence of the initial conditions. In the special case $a = 4$, it turns out that replacing x by $\text{Sin}[\pi u]^2$ makes the mapping become just $u \to \text{FractionalPart}[2u]$, revealing simple shift map dependence on the initial digit sequence. (See pages 1090 and 1098.)



■ **Higher-dimensional generalizations.** One can consider so-called Anosov maps such as $\{x, y\} \to \text{Mod}[m . \{x, y\}, 1]$ where m is a matrix such as $\{\{2, 1\}, \{1, 1\}\}$. Any initial condition containing only rational numbers will then yield repetitive behavior, much as in the shift map. But as soon as m itself contains rational numbers, complicated behavior can be obtained even with an initial condition such as $\{1, 1\}$.

■ **Distribution of chaotic behavior.** For iterated maps, unlike for discrete systems such as cellular automata, one can get continuous ranges of rules by varying parameters. With maps based on piecewise linear functions the regions of parameters in which chaotic behavior occurs typically have simple shapes; with maps based, say, on quadratic functions, however, elaborate nested shapes can occur. (See page 934.)

■ **Page 155 · Lyapunov exponents.** The number of new digits that are affected at each step by a small change in initial conditions gives the so-called Lyapunov exponent λ for the evolution. After t steps, the difference in size resulting from the change in initial conditions will be multiplied by approximately $2^{\lambda t}$—at least until this difference is of order 1. (See page 950.)

■ **Chaos in nature.** See page 304.

■ **Bitwise operations.** Cellular automata can be thought of as analogs of iterated maps in which bitwise operations such as *BitXor* are used instead of ordinary arithmetic ones. (See page 906.)

Continuous Cellular Automata

■ **Implementation.** The state of a continuous cellular automaton at a particular step can be represented by a list of numbers, each lying between 0 and 1. This list can then be updated using

```
CCAEvolveStep[f_, list_List] :=
  Map[f, (RotateLeft[list] + list + RotateRight[list])/3]
CCAEvolveList[f_, init_List, t_Integer] :=
  NestList[CCAEvolveStep[f, #] &, init, t]
```

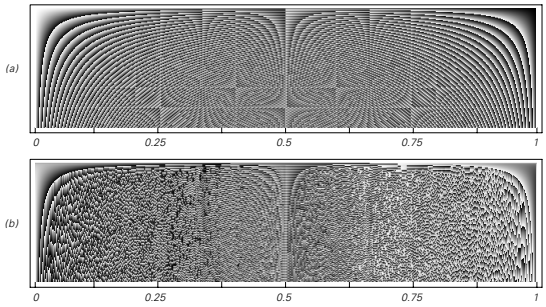
where for the rule on page 157 f is *FractionalPart[3#/2] &* while for the rule on page 158 it is *FractionalPart[# + 1/4] &*.

Note that in the definitions above, the elements of *list* can be either exact rational numbers, or approximate numbers obtained using N. For rough calculations, standard machine-precision numbers may sometimes suffice, but for detailed calculations exact rational numbers are essential. Indeed, the presence of exponentially increasing errors would make the bottom of the picture on page 157 qualitatively wrong if just 64-bit double-precision numbers had been used. On page 160 the effect is much larger, and almost all the pictures would be completely wrong—with the notable exception of the one that shows localized structures.

■ **History.** Continuous cellular automata have been introduced independently several times, under several different names. In all cases the rules have been at least slightly more complicated than the ones I consider here, and behavior starting from simple initial conditions does not appear to have been studied before. Versions of continuous cellular automata arose in the mid-1970s as idealizations of coupled ordinary differential equations for arrays of nonlinear oscillators, and implicitly in finite difference approximations to partial differential equations. They began

to be studied with extensive computer simulations in the early 1980s, probably following my work on ordinary cellular automata. Most often considered, notably by Kunihiko Kaneko and co-workers, were so-called "coupled map lattices" or "lattice dynamical systems" in which an iterated map (typically a logistic map) was applied at each step to a combination of neighboring cell value. A transition from regular class 2 to irregular class 3 behavior, with class 4 behavior involving localized structures in between, was observed, and was studied in detail by Hugues Chaté and Paul Manneville, starting in the late 1980s.

■ **Page 158 · Properties.** At step t the background is *FractionalPart[a t]*. For rational a this always repeats, cycling through *Denominator[a]* possible values (compare page 255). In most patterns generated from initial conditions containing say a single black cell most cells whose values are not forced to be the same end up being at least slightly different—even in cases like $a = 0.375$. Note that in cases like $a = 0.475$ there is some trace of a pattern at every step—but it only becomes obvious when it makes values wrap around from 1 to 0. The pictures below show successive colors of (a) the background (compare page 950) and (b) the center cell for each $a = n/500$ from 0 to 1 for the systems on page 159. (Compare page 243.)

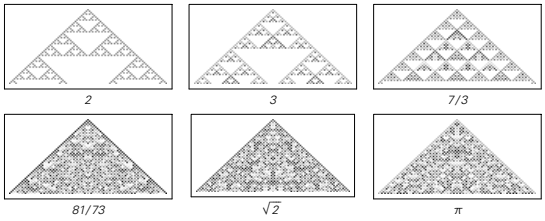


If a is not a rational number the background never repeats, but the main features of patterns obtained seem similar.

■ **Additive rules.** In the case $a = 0$ the systems on page 159 are purely additive. A simpler example is the rule

Mod[RotateLeft[list] + RotateRight[list], 1]

With a single nonzero initial cell with value $1/k$ the pattern produced is just Pascal's triangle modulo k. If k is a rational number only a limited set of values appear, and the pattern has a nested form analogous to those shown on page 870. If k is irrational then equidistribution of *Mod[Binomial[t, x], k]* implies that all possible values eventually appear; the corresponding patterns seem fairly irregular, as shown below. (Compare pages 953 and 1092.)



■ **Probabilistic cellular automata.** As an alternative to having continuous values at each cell, one can consider ordinary cellular automata with discrete values, but introduce probabilities for, say, two different rules to be applied at each cell. Examples of probabilistic cellular automata are shown on page 591; their behavior is typically quite similar to continuous cellular automata.

Partial Differential Equations

■ **Ordinary differential equations.** It is also possible to set up systems which have a finite number of continuous variables (say *a[t]*, *b[t]*, etc.) that change continuously with time. The rules for such systems correspond to ordinary differential equations. Over the past century, the field of dynamical systems theory has produced many results about such systems. If all equations are of the form *a'[t] == f[a[t], b[t], …]*, etc. then it is known for example that it is necessary to have at least three equations in order to get behavior that is not ultimately fixed or repetitive. (The Lorenz equations are an example.) If the function f depends explicitly on time, then two equations suffice. (The van der Pol equations are an example.)

Just as in iterated maps, a small change in the initial values *a[0]* etc. can often lead to an exponentially increasing difference in later values of *a[t]*, etc. But as in iterated maps, the main part of this process that has been analyzed is simply the excavation of progressively less significant digits in the number *a[0]*.

(Note that numerical simulations of ODEs on computers must approximate continuous time by discrete steps, making the system essentially an iterated map, and often yielding spurious complicated behavior.)

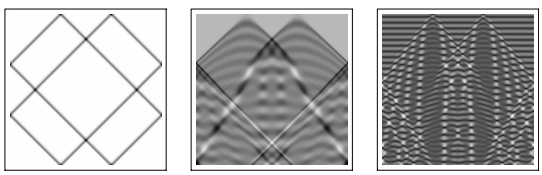
■ **Klein-Gordon equation.** The behavior of the Klein-Gordon equation $\partial_{tt} u[t, x] == \partial_{xx} u[t, x] - u[t, x]$ is visually very similar to that shown for the sine-Gordon equation. For the Klein-Gordon equation, however, there is an exact solution:

$u[t, x] = If[x^2 > t^2, 0, BesselJ[0, Sqrt[t^2 - x^2]]]$

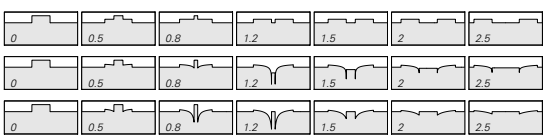
■ **Origins of the equations.** The diffusion equation arises in physics from the evolution of temperature or of gas density.

The wave equation represents the propagation of linear waves, for example along a compressible spring. The sine-Gordon equation represents nonlinear waves obtained for example as the limit of a very large number of pendulums all connected to a spring. The traditional name of the equation is a pun on the Klein-Gordon equation that appears in relativistic quantum mechanics and in describing strings in elastic media. It is notable that unlike with ODEs, essentially all PDEs that have been widely studied come quite directly from physics. My PDE on page 165 is however an exception.

■ **Nonlinearity.** The pictures below show behavior with initial conditions containing two Gaussians (and periodic boundary conditions). The diffusion and wave equations are linear, so that results are linear sums of those with single Gaussians. The sine-Gordon equation is nonlinear, but its solutions satisfy a generalized linear superposition principle. The equation from page 165 shows no such simple superposition principle. Note that even with a linear equation, fairly complicated patterns of behavior can sometimes emerge as a result of boundary conditions.


wave equation sine-Gordon equation my equation

■ **Higher dimensions.** The pictures below show as examples the solution to the wave equation in 1D, 2D and 3D starting from a stationary square pulse.



In each case a 1D slice through the solution is shown, and the solution is multiplied by r^{d-1}. For the wave equation, and for a fair number of other equations, even and odd dimensions behave differently. In 1D and 3D, the value at the origin quickly becomes exactly 0; in 2D it is given by $1 - t/\text{Sqrt}[t^2 - 1]$, which tends to zero only like $-1/(2\,t^2)$ (which means that a sound pulse cannot propagate in a normal way in 2D).

■ **Page 164 · Singular behavior.** An example of an equation that yields inconsistent behavior is the diffusion equation with a negative diffusion constant:

$\partial_t u[t, x] == -\partial_{xx} u[t, x]$

This equation makes any variation in u as a function of x eventually become infinitely rapid.

Many equations used in physics can lead to singularities: the Navier-Stokes equations for fluid flow yield shock waves, while the Einstein equations yield black holes. At a physical level, such singularities usually indicate that processes not captured by the equations have become important. But at a mathematical level one can simply ask whether a particular equation always has solutions which are at least as regular as its initial conditions. Despite much work, however, only a few results along these lines are known.

■ **Existence and uniqueness.** Unlike systems such as cellular automata, PDEs do not have a built-in notion of "evolution" or "time". Instead, as discussed on page 940, a PDE is essentially just a constraint on the values of a function at different times or different positions. In solving a PDE, one is usually interested in determining values that satisfy this constraint inside a particular region, based on information about values on the edges. It is then a fundamental question how much can be specified on the edges in order to obtain a unique solution. If too little is specified, there may be many possible solutions, while if too much is specified there may be no consistent solution at all. For some very simple PDEs, the conditions for unique solutions are known. So-called hyperbolic equations (such as the wave equation, the sine-Gordon equation and my equation) work a little like cellular automata in that in at least one dimension information can propagate only at a limited speed, say c. The result is that in such equations, giving values for $u[t, x]$ at $t = 0$ for $-s < x < s$ will uniquely determine $u[t, x]$ at larger t for $-s + c\,t < x < s - c\,t$. In other PDEs, such as so-called elliptic ones, there is no such limit on the rate of information propagation, and as a result, it is immediately necessary to know values of $u[t, x]$ at all x, and on the boundaries of the region, in order to determine $u[t, x]$ for any $t > 0$.

■ **Page 165 · Field equations.** Any equation of the form

$\partial_{tt} u[t, x] == \partial_{xx} u[t, x] + f[u[t, x]]$

can be thought of as a classical field equation for a scalar field. Defining

$v[u] = -\text{Integrate}[f[u], u]$

the field then has Lagrangian density

$((\partial_t u)^2 - (\partial_x u)^2)/2 - v[u]$

and conserves the Hamiltonian (energy function)

$\text{Integrate}[((\partial_t u)^2 + (\partial_x u)^2)/2 + v[u], \{x, -\infty, \infty\}]$

With the choice for $f[u]$ made here (with $a \geq 0$), $v[u]$ is bounded from below, and as a result it follows that no singularities ever occur in $u[t, x]$.

■ **Equation for the background.** If $u[t, x]$ is independent of x, as it is sufficiently far away from the main pattern, then the partial differential equation on page 165 reduces to the ordinary differential equation

$$u''[t] == (1 - u[t]^2)(1 + a\,u[t])$$

$$u[0] == u'[0] == 0$$

For $a = 0$, the solution to this equation can be written in terms of Jacobi elliptic functions as

$$\sqrt{3}\ \text{JacobiSN}[t/3^{1/4}, 1/2]^2/(1 + \text{JacobiCN}[t/3^{1/4}, 1/2]^2)$$

In general the solution is

$$b\,d\,\text{JacobiSN}[r\,t, s]^2/(b - d\,\text{JacobiCN}[r\,t, s]^2)$$

where

$$r = -\text{Sqrt}[1/8\,a\,c\,(b - d)]$$

$$s = d\,(c - b)/(c\,(d - b))$$

and b, c, d are determined by the equation

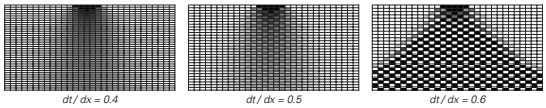
$$(x - b)(x - c)(x - d) == -(12 + 6\,a\,x - 4\,x^2 - 3\,a\,x^3)/(3\,a)$$

In all cases (except when $-8/3 < a < -1/\sqrt{6}$), the solution is periodic and non-singular. For $a = 0$, the period is $2\,3^{1/4}\,\text{EllipticK}[1/2] \simeq 4.88$. For $a = 1$, the period is about 4.01; for $a = 2$, it is about 3.62; while for $a = 4$, it is about 3.18. For $a = 8/3$, the solution can be written without Jacobi elliptic functions, and is given by

$$3\,\text{Sin}[\text{Sqrt}[5/6]\,t]^2/(2 + 3\,\text{Cos}[\text{Sqrt}[5/6]\,t]^2)$$

■ **Numerical analysis.** To find numerical solutions to PDEs on a digital computer one has no choice but to make approximations. In the typical case of the finite difference method one sets up a system with discrete cells in space and time that is much like a continuous cellular automaton, and then hopes that when the cells in this system are made small enough its behavior will be close to that of the continuous PDE.

Several things can go wrong, however. The pictures below show as one example what happens with the diffusion equation when the cells have size dt in time and dx in space. So long as the so-called Courant condition $dt/dx < 1/2$ is satisfied, the results are correct. But when dt/dx is made larger, an instability develops, and the discrete approximation yields completely different results from the continuous PDE.



$dt/dx = 0.4$ $dt/dx = 0.5$ $dt/dx = 0.6$

Many methods beyond finite differences have been invented over the past 30 years for finding numerical solutions to PDEs. All however ultimately involve discretization, and can suffer from difficulties that are similar—though often more insidious—to those for finite differences.

For equations where one can come at least close to having explicit algebraic formulas for solutions, it has often been possible to prove that a certain discretization procedure will yield correct results. But when the form of the true solution is more complicated, such proofs are typically impossible.

And indeed in practice it is often difficult to tell whether complexity that is seen is actually a consequence of the underlying PDE, or is instead merely a reflection of the discretization procedure. I strongly suspect that many equations, particularly in fluid dynamics, that have been studied over the past few decades exhibit highly complex behavior. But in most publications such behavior is never shown, presumably because the authors are not sure whether the behavior is a genuine consequence of the equations they are studying.

■ **Implementation.** All the numerical solutions shown were found using the *NDSolve* function built into *Mathematica*. In general, finite difference methods, the method of lines and pseudospectral methods can be used. For equations of the form

$$\partial_{tt}\,u[t, x] == \partial_{xx}\,u[t, x] + f[u[t, x]]$$

one can set up a simple finite difference method by taking f in the form of pure function and creating from it a kernel with space step dx and time step dt:

```
PDEKernel[f_, {dx_, dt_}] := Compile[{a, b, c, d},
    Evaluate[(2 b - d) + ((a + c - 2 b)/dx^2 + f[b]) dt^2]]
```

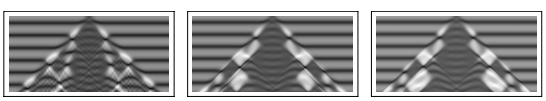
Iteration for n steps is then performed by

```
PDEEvolveList[ker_, {u0_, u1_}, n_] :=
  Map[First, NestList[PDEStep[ker, #] &, {u0, u1}, n]]
PDEStep[ker_, {u1_, u2_}] := {u2, Apply[ker, Transpose[
      {RotateLeft[u2], u2, RotateRight[u2], u1}], {1}]}
```

With this approach an approximation to the top example on page 165 can be obtained from

```
PDEEvolveList[PDEKernel[
    (1 - #^2) (1 + #) &, {0.1, 0.05}], Transpose[
    Table[{1, 1} N[Exp[-x^2]], {x, -20, 20, 0.1}]], 400]
```

For both this example and the middle one the results converge rapidly as dx decreases. But for the bottom example, the pictures below show that convergence is not so rapid, and indeed, as is typical in working with PDEs, despite having used large amounts of computer time I do not know whether the details of the picture in the main text are really correct. The energy function (see above) is at least roughly conserved, but it seems quite likely that the "shocks" visible are merely a consequence of the discretization procedure used.

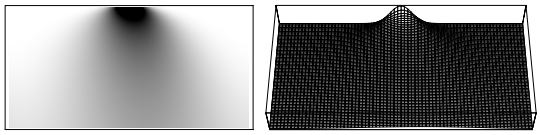


$dx = 0.5$ $dx = 0.2$ $dx = 0.05$

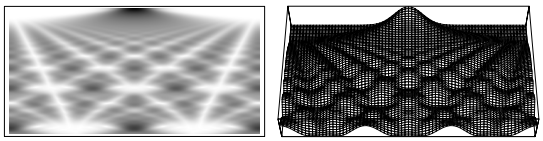
■ **Different powers.** The equations

$\partial_{tt} u[t, x] == \partial_{xx} u[t, x] + (1 - u[t, x]^n)(1 + a\, u[t, x])$

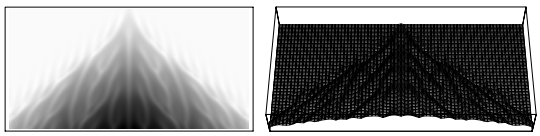
with $n = 4, 6, 8$, etc. appear to show similar behavior to the $n = 2$ equation in the main text.



Burger's equation: $\partial_t u[t, x] = \partial_{xx} u[t, x] - u[t, x]\, \partial_x u[t, x]$



nonlinear Schrödinger equation: $i\, \partial_t u[t, x] = -\partial_{xx} u[t, x] + 4\, \text{Abs}[u[t, x]]^2\, u[t, x]$



Kuramoto-Sivashinsky equation: $\partial_t u[t, x] = -\partial_{xx} u[t, x] - 1/2\, \partial_{xxxx} u[t, x] + (\partial_x u[t, x])^2$

■ **Other PDEs.** The pictures above show three PDEs that have been studied in recent years. All are of the so-called parabolic type, so that, unlike my equation, they have no limit on the rate of information propagation, and thus a solution in any region immediately depends on values on the boundary—which in the pictures below is taken to be periodic. (The deterministic Kardar-Parisi-Zhang equation $\partial_t u[t, x] == a\, \partial_{xx} u[t, x] + 1/2\, b\, (\partial_x u[t, x])^2$ yields behavior like Burger's equation, but symmetrical. Note that *Abs[u]* is plotted in the second picture, while for the last equation a common less symmetrical form replaces the last term by $u[t, x]\, \partial_x u[t, x]$.)

Continuous Versus Discrete Systems

■ **History.** From the late 1600s when calculus was invented it took about two centuries before mathematicians came to terms with the concepts of continuity that it required. And to do so it was necessary to abandon concrete intuition, and instead to rely on abstract mathematical theorems. (See page 1149.) The kind of discrete systems that I consider in this book allow a return to a more concrete form of mathematics, without the necessity for such abstraction.

■ **"Calculus".** It is an irony of language that the word "calculus" now associated with continuous systems comes from the Latin word which means a small pebble of the kind used for doing discrete calculations (same root as "calcium").

Two Dimensions and Beyond

Introduction

▪ **Other lattices.** See page 929.

▪ **Page 170 · 1D phenomena.** Among the phenomena that cannot occur in one dimension are those associated with shape, winding and knotting, as well as traditional phase transitions with reversible evolution rules (see page 981).

Cellular Automata

▪ **Implementation.** An $n \times n$ array of white squares with a single black square in the middle can be generated by

```
PadLeft[{{1}}, {n, n}, 0, Floor[{n, n}/2]]
```

For the 5-neighbor rules introduced on page 170 each step can be implemented by

```
CAStep[rule_, a_] := Map[rule[[10 - #]] &,
    ListConvolve[{{0, 2, 0}, {2, 1, 2}, {0, 2, 0}}, a, 2], {2}]
```

where *rule* is obtained from the *code* number by *IntegerDigits[code, 2, 10]*.

For the 9-neighbor rules introduced on page 177

```
CAStep[rule_, a_] := Map[rule[[18 - #]] &,
    ListConvolve[{{2, 2, 2}, {2, 1, 2}, {2, 2, 2}}, a, 2], {2}]
```

where *rule* is given by *IntegerDigits[code, 2, 18]*.

In d dimensions with k colors, 5-neighbor rules generalize to $(2d+1)$-neighbor rules, with

```
CAStep[{rule_, d_}, a_] :=
  Map[rule[[-1 - #]] &, a + k ListConvolve[a, d], {d}]
ListConvolve[a_, d_] := Apply[Plus, Map[RotateLeft[a, #] +
      RotateRight[a, #] &, IdentityMatrix[d]]]
```

with *rule* given by *IntegerDigits[code, k, k (2 d (k - 1) + 1)]*.

9-neighbor rules generalize to 3^d-neighbor rules, with

```
CAStep[{rule_, d_}, a_] :=
  Map[rule[[-1 - #]] &, a + k FullTotal[a, d], {d}]
FullTotal[a_, d_] :=
  Array[RotateLeft[a, {##}] &, Table[3, {d}], -1, Plus] - a
```

with *rule* given by *IntegerDigits[code, k, k ((3^d - 1) (k - 1) + 1)]*.

In 3 dimensions, the positions of black cells can conveniently be displayed using

```
Graphics3D[Map[Cuboid[-Reverse[#]] &, Position[a, 1]]]
```

▪ **General rules.** One can specify the neighborhood for any rule in any dimension by giving a list of the offsets for the cells used to update a given cell. For 1D elementary rules the list is *{{-1}, {0}, {1}}*, while for 2D 5-neighbor rules it is *{{-1, 0}, {0, -1}, {0, 0}, {0, 1}, {1, 0}}*. In this book such offset lists are always taken to be in the order given by *Sort*, so that for range r rules in d dimensions the order is the same as *Flatten[Array[List, Table[2 r + 1, {d}], -r], d - 1]*. One can specify a neighborhood configuration by giving in the same order as the offset list the color of each cell in the neighborhood. With offset list *os* and k colors the possible neighborhood configurations are

```
Reverse[Table[IntegerDigits[i - 1,
    k, Length[os]], {i, k ^ Length[os]}]]
```

(These are shown on page 53 for elementary rules and page 941 for 5-neighbor rules.) If a cellular automaton rule takes the new color of a cell with neighborhood configuration *IntegerDigits[i, k, Length[os]]* to be *u[[i + 1]]*, then one can define its rule number to be *FromDigits[Reverse[u], k]*. A single step in evolution of a general cellular automaton with state a and rule number *num* is then given by

```
Map[IntegerDigits[num, k, k ^ Length[os]][[-1 - #]] &,
  Apply[Plus, MapIndexed[k ^ (Length[os] - First[#2])
      RotateLeft[a, #1] &, os]], {-1}]
```

or equivalently by

```
Map[IntegerDigits[num, k, k ^ Length[os]][[-# - 1]] &,
  ListCorrelate[Fold[ReplacePart[k #1, 1, #2 + r + 1] &,
    Array[0 &, Table[2 r + 1, {d}]], os], a, r + 1], {d}]
```

▪ **Numbers of possible rules.** The table below gives the total number of 2D rules of various types with two possible colors for each cell. Given an initial pattern with a certain symmetry, a rule will maintain that symmetry if the rule is such that every neighborhood equivalent under the symmetry yields the same color of cell. Rules are considered rotationally

symmetric in the table below if they preserve any possible rotational symmetry consistent with the underlying arrangement of cells. Totalistic rules depend only on the total number of black cells in a neighborhood; outer totalistic rules (as in the previous note) also depend on the color of the center cell. Growth totalistic rules make any cell that becomes black remain black forever.

In such a rule, given a list of how many neighbors around a given cell (out of s possible) make the cell turn black the outer totalistic code for the rule can be obtained from

```
Apply[Plus, 2^Join[2 list, 2 Range[s + 1] - 1]]
```

	5 - neighbor square	9 - neighbor square	hexagonal
general	$2^{32} \approx 4 \times 10^{9}$	$2^{512} \approx 10^{154}$	$2^{128} \approx 3 \times 10^{38}$
rotationally symmetric	$2^{12} = 4096$	$2^{140} \approx 10^{42}$	$2^{28} \approx 3 \times 10^{8}$
completely symmetric	$2^{12} = 4096$	$2^{102} \approx 5 \times 10^{30}$	$2^{26} \approx 7 \times 10^{7}$
outer totalistic	$2^{10} = 1024$	$2^{18} \approx 3 \times 10^{5}$	$2^{14} = 16384$
totalistic	$2^{6} = 64$	$2^{10} = 1024$	$2^{8} = 256$
growth totalistic	$2^{5} = 32$	$2^{9} = 512$	$2^{7} = 128$

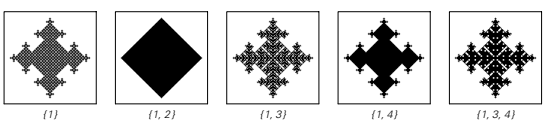
■ **Symmetric 5-neighbor rules.** Among the 32 possible 5-cell neighborhoods shown for example on page 941 there are 12 classes related by symmetries, given by

```
s = {{1}, {2, 3, 9, 17}, {4, 10, 19, 25},
     {5}, {6, 7, 13, 21}, {8, 14, 23, 29}, {11, 18},
     {12, 20, 26, 27}, {15, 22}, {16, 24, 30, 31}, {28}, {32}}
```

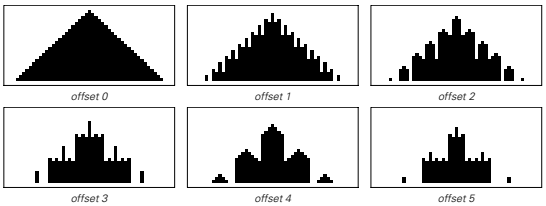
Completely symmetric 5-neighbor rules can be numbered from 0 to 4095, with each digit specifying the new color of the cell for each of these symmetry classes of neighborhoods. Such rule numbers can be converted to general form using

```
FromDigits[Map[Last, Sort[Flatten[Map[Thread,
      Thread[{s, IntegerDigits[n, 2, 12]}]], 1]]], 2]
```

■ **Growth rules.** The pictures below show examples of rules in which a cell becomes black if it has exactly the specified numbers of black neighbors (the initial conditions used have the minimal number of black cells for growth). The code numbers in these cases are given by $2/3\,(4^n - 1) + \text{Apply[Plus, } 4^{\text{list}}\text{]}$ where n is the number of neighbors, here 5. (See also the 9-neighbor examples on page 373.)



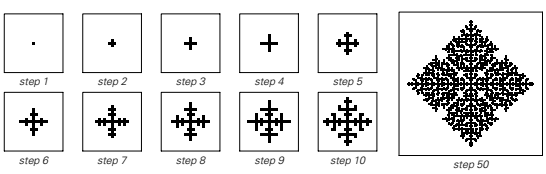
■ **Page 171 · Code 942 slices.** The following is the result of taking vertical slices through the pattern with a sequence of offsets from the center:



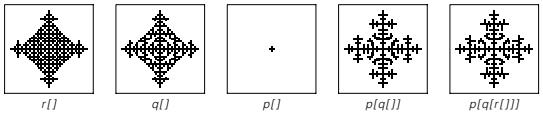
■ **History.** As indicated on pages 876–878, 2D cellular automata were historically studied more extensively than 1D ones—though rarely with simple initial conditions. The 5-cell neighborhood on page 170 was considered by John von Neumann in 1952; the 9-cell one on page 177 by Edward Moore in 1962. (Both are also common in finite difference approximations in numerical analysis.) (The 7-cell hexagonal neighborhood of page 369 was considered for image processing purposes by Marcel Golay in 1959.) Ever since the invention of the Game of Life around 1970 a remarkable number of hardware and software simulators have been built to watch its evolution. But until after my work in the 1980s simulators for more general 2D cellular automata were rare. A sequence of hardware simulators were nevertheless built starting in the mid-1970s by Tommaso Toffoli and later Norman Margolus. And as mentioned on page 1077, going back to the 1950s some image processing systems have been based on particular families of 2D cellular automaton rules.

■ **Ulam systems.** Having formulated the system around 1960, Stanislaw Ulam and collaborators (see page 877) in 1967 simulated 120 steps of the process shown below, with black cells after t steps occurring at positions

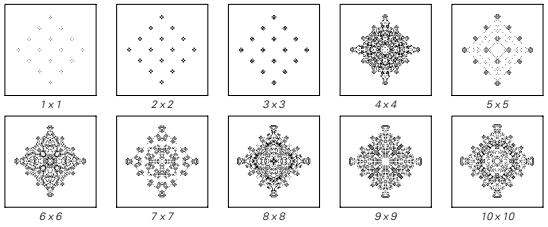
```
Map[First,
  First[Nest[UStep[p[q[r[#1], #2]] &, {{1, 0}, {0, 1}, {-1, 0},
       {0, -1}}, #] &, ({#, #} &)[{{{0, 0}, {0, 0}}}], t]]]

UStep[f_, os_, {a_, b_}] := ({Join[a, #], #} &)[f[Flatten[
      Outer[{#1 + #2, #1} &, Map[First, b], os, 1], 1], a]]

r[c_] := Map[First, Select[Split[Sort[c],
      First[#1] == First[#2] &], Length[#] == 1 &]]

q[c_, a_] := Select[c,
    Apply[And, Map[Function[u, qq[#1, u, a]], a]] &]

p[c_] := Select[c,
    Apply[And, Map[Function[u, pp[#1, u]], c]] &]

pp[{x_, u_}, {y_, v_}] := Max[Abs[x - y]] > 1 || u == v

qq[{x_, u_}, {y_, v_}, a_] := x == y || Max[Abs[x - y]] > 1 ||
    u == y || First[Cases[a, {u, z_} → z]] == y
```



These rules are fairly complicated, and involve more history than ordinary cellular automata. But from the discoveries in this book we now know that much simpler rules can also yield very complicated behavior. And as the pictures below show, this is true even just for parts of the rules above (s alone yields outer totalistic code 686 in 2D, and rule 90 in 1D).



Ulam also in 1967 considered the pure 2D cellular automaton with outer totalistic code 12 (though he stated its rule in a complicated way). As shown in the pictures below, when started from blocks of certain sizes this rule yields complex patterns—although nothing like this was noted in 1967.



▪ **Limiting shapes.** When growth occurs at the maximum rate the outer boundaries of a cellular automaton pattern reflect the neighborhood involved in its underlying rule (in rough analogy to the Wulff construction for shapes of crystals). When growth occurs at a slower rate, a wide range of polygonal and other shapes can be obtained, as illustrated in the main text.

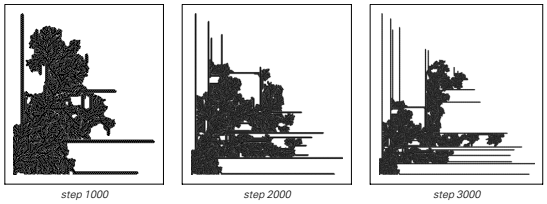
▪ **Additive rules.** See page 1092.

▪ **Page 174 · Cellular automaton art.** 2D cellular automata can be used to make a wide range of designs for rugs, wallpaper, and similar objects. Repeating squares of pattern can be produced by using periodic boundary conditions. Rules with more than two colors will sometimes be appropriate. For rugs, it is typically desirable to have each cell correspond to more than one tuft, since otherwise with most rules the rug looks too busy. (Compare page 872.)

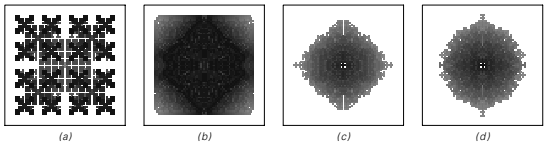
▪ **Page 177 · Code 175850.** See also page 980.

▪ **Page 178 · Code 746.** The pattern generated is not perfectly circular, as discussed on page 979. Its interior is mostly fixed, but there are scattered small regions that cycle with a variety of periods.

▪ **Page 181 · Code 174826.** The pictures below show the upper-right quadrant for more steps. Most of the lines visible are 8 cells across, and grow by 4 cells every 12 steps. They typically survive being hit by more complicated growth from the side. But occasionally runners 3 cells wide will start on the side of a line. And since these go 2 cells every 3 steps they always catch up with lines, producing complicated growth, often terminating the lines.



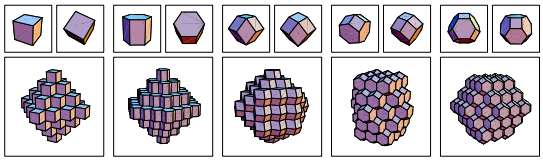
▪ **Page 183 · Projections from 3D.** Looking from above, with closer cells shown darker, the following show patterns generated after 30 steps, by (a) the rule at the top of page 183, (b) the rule at the bottom of page 183, (c) the rule where a cell becomes black if exactly 3 out of 26 neighbors were black and (d) the same as (c), but with a 3×3×1 rather than a 3×1×1 initial block of black cells:



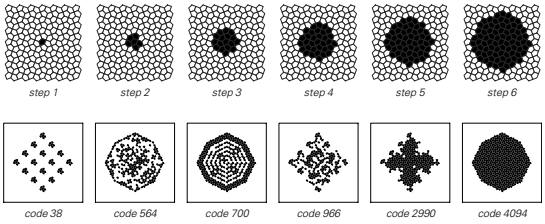
▪ **Other geometries.** Systems like cellular automata can readily be set up on any geometrical structure in which a limited number of types of cells can be identified, with every cell of a given type having a similar neighborhood.

In the simplest case, the cells are all identical, and are laid out in the same orientation in a repetitive array. The centers of the cells form a lattice, with coordinates that are integer multiples of some set of basis vectors. The possible complete symmetries of such lattices are much studied in crystallography. But for the purpose of nearest-neighbor cellular automaton rules, what matters is not detailed geometry, but merely what cells are adjacent to a given cell. This can be determined by looking at the Voronoi region (see page 987) for each point in the lattice. In any given dimension, this region (variously known as a Dirichlet domain or Wigner-Seitz cell, and dual to the primitive cell, first Brillouin zone or Wulff shape) has a limited number of possible overall shapes. The most symmetrical versions of these shapes in 2D are the square (4 neighbors) and hexagon (6) and in 3D (as found by Evgraf Fedorov in 1885) the cube (6), hexagonal prism (8), rhombic dodecahedron (12) (e.g.

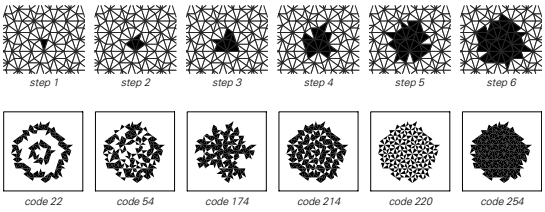
face-centered cubic crystals), rhombo-hexagonal or elongated dodecahedron (12) and truncated octahedron or tetradecahedron (14) (e.g. body-centered cubic crystals), as shown below. (In 4D, 8, 16 and 24 nearest neighbors are possible; in higher dimensions possibilities have been investigated in connection with sphere packing.) (Compare pages 1029 and 986.)



In general, there is no need for individual cells in a cellular automaton to have the same orientation. A triangular lattice is one example where they do not. And indeed, any tiling of congruent figures can readily be used to make a cellular automaton, as illustrated by the pentagonal example below. (Outer totalistic codes specify rules; the first rule makes a particular cell black when any of its five neighbors are black and has code 4094. Note that even though individual cells are pentagonal, large-scale cellular automaton patterns usually have 2-, 4- or 8-fold symmetry.)



There is even no need for the tiling to be repetitive; the picture below shows a cellular automaton on a nested Penrose tiling (see page 932). This tiling has two different shapes of tile, but here both are treated the same by the cellular automaton rule, which is given by an outer totalistic code number. The first example is code 254, which makes a particular cell become black when any of its three neighbors are black. (Large-scale cellular automaton patterns here can have 5-fold symmetry.) (See also page 1027.)



▪ **Networks.** Cellular automata can be set up so that each cell corresponds to a node in a network. (See page 936.) The only requirement is that around each node the network must have the same structure (or at least a limited number of possible structures). For nearest-neighbor rules, it suffices that each node has the same number of connections. For longer-range rules, the network must satisfy constraints of the kind discussed on page 483. (Cayley graphs of groups always have the necessary homogeneity.) If the connections at each node are not labelled, then only totalistic cellular automaton rules can be implemented. Many topological and geometrical properties of the underlying network can affect the overall behavior of a cellular automaton on it.

Turing Machines

▪ **Implementation.** With rules represented as a list of elements of the form *{s, a}→{sp, ap, {dx, dy}}* (*s* is the state of the head and *a* the color of the cell under the head) each step in the evolution of a 2D Turing machine is given by

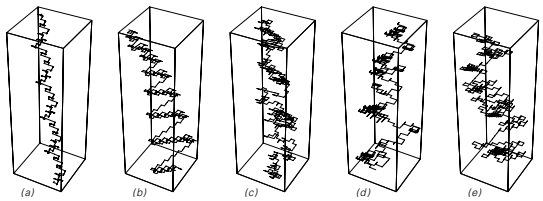
```
TM2DStep[rule_, {s_, tape_, r : {x_, y_}}] :=
  Apply[{#1, ReplacePart[tape, #2, {r}], r + #3} &,
    {s, tape[[x, y]]} /. rule]
```

▪ **History.** At a formal level 2D Turing machines have been studied since at least the 1950s. And on several occasions systems equivalent to specific simple 2D Turing machines have also been constructed. In fact, much as for cellular automata, more explicit experiments have been done on 2D Turing machines than 1D ones. A tradition of early robotics going back to the 1940s—and leading for example to the Logo computer language—involved studying idealizations of mobile turtles. And in 1971 Michael Paterson and John Conway constructed what they described as an idealization of a prehistoric worm, which was essentially a 2D Turing machine in which the state of the head records the direction of the motion taken at each step. Michael Beeler in 1973 used a computer at MIT to investigate all 1296 possible worms with rules of the simplest type on a hexagonal grid, and he found several with fairly complex behavior. But this discovery does not appear to have been followed up, and systems equivalent to simple 2D Turing machines were reinvented again, largely independently, several times in the mid-1980s: by Christopher Langton in 1985 under the name "vants"; by Rudy Rucker in 1987 under the name "turmites"; and by Allen Brady in 1987 under the name "turning machines". The specific 4-state rule

```
{s_, c_} :> With[{sp = s (2 c - 1) ii},
    {sp, 1 - c, {Re[sp], Im[sp]}}]
```

has been called Langton's ant, and various studies of it were done in the 1990s.

■ **Visualization.** The pictures below show the 2D position of the head at 500 successive steps for the rules on page 185.



Some 2D Turing machines exhibit elements of randomness at some steps, but then fill in every so often to form simple repetitive patterns. An example is the 3-state rule

■ **Rules based on turning.** The rules used in the main text specify the displacement of the head at each step in terms of fixed directions in the underlying grid. An alternative is to specify the turns to make at each step in the motion of the head. This is how turtles in the Logo computer language are set up. (Compare the discussion of paths in substitution systems on page 892.)

■ **2D mobile automata.** Mobile automata can be generalized just like Turing machines. Even in the simplest case, however, with only four neighbors involved there are already $(4k)^{k^5}$ possible rules, or nearly 10^{29} even for $k = 2$.

Substitution Systems and Fractals

■ **Implementation.** With the rule on page 187 given for example by *{1 → {{1, 0}, {1, 1}}, 0 → {{0, 0}, {0, 0}}}* the result of *t* steps in the evolution of a 2D substitution system from a initial condition such as *{{1}}* is given by

```
SS2DEvolve[rule_, init_, t_] :=
  Nest[Flatten2D[# /. rule] &, init, t]
Flatten2D[list_] :=
  Apply[Join, Map[MapThread[Join, #] &, list]]
```

■ **Connection with digit sequences.** Just as in the 1D case discussed on page 891, the color of a cell at position *{i, j}* in a 2D substitution system can be determined using a finite automaton from the digit sequences of the numbers *i* and *j*. At step *n*, the complete array of cells is

```
Table[If[FreeQ[Transpose[IntegerDigits[{i, j}, k, n]], form],
    1, 0], {i, 0, k^n - 1}, {j, 0, k^n - 1}]
```

where for the pattern on page 187, $k = 2$ and $form = \{0, 1\}$. For patterns (a) through (f) on page 188, $k = 3$ and *form* is given respectively by (a) *{1, 1}*, (b) *{0 | 2, 0 | 2}*, (c) *{0 | 2, 0 | 2} | {1, 1}*, (d) *{i_, j_} /; j > i*, (e) *{0, 2} | {1, 1} | {2, 0}*, (f) *{0, 2} | {1, 1}*. Note that the excluded pairs of digits are in exact correspondence with the positions of which squares are *0* in the underlying rules for the substitution systems. (See pages 608 and 1091.)

■ **Page 187 · Sierpiński pattern.** Other ways to generate step *n* of the pattern shown here in various orientations include:

- *Mod[Array[Binomial, {2, 2}^n, 0], 2]* (see pages 611 and 870)
- *1 - Sign[Array[BitAnd, {2, 2}^n, 0]]* (see pages 608 and 871)
- *NestList[Mod[RotateLeft[# + #, 2] &, PadLeft[{1}, 2^n], 2^n - 1]* (see page 870)
- *NestList[Mod[ListConvolve[{1, 1}, #, -1], 2] &, PadLeft[{1}, 2^n], 2^n - 1]* (see page 870)
- *IntegerDigits[NestList[BitXor[2 #, #] &, 1, 2^n - 1], 2, 2^n]* (see page 906)
- *NestList[Mod[Rest[FoldList[Plus, 0, #]], 2] &, Table[1, {2^n}], 2^n - 1]* (see page 1034)
- *Table[PadRight[Mod[CoefficientList[(1 + x)^(t-1), x], 2], 2^n - 1], {t, 2^n}]* (see pages 870 and 951)
- *Reverse[Mod[CoefficientList[Series[1/(1 - (1 + x) y), {x, 0, 2^n - 1}, {y, 0, 2^n - 1}], {x, y}], 2]]* (see page 1091)
- *Nest[Apply[Join, MapThread[Join, {{#, #}, {0#, #}}, 2]] &, {{1}}, n]* (compare page 1073)

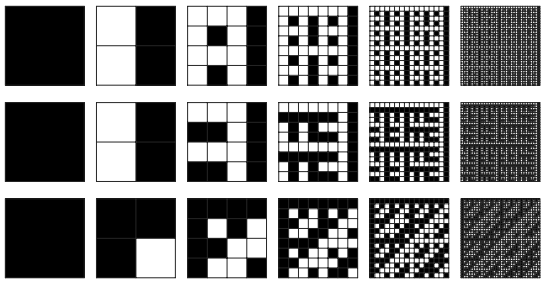
The positions of black squares can be found from:

- *Nest[Flatten[2 # /. {x_, y_} → {{x, y}, {x + 1, y}, {x, y + 1}}, 1] &, {{0, 0}}, n]*
- *(Transpose[{Re[#], Im[#]}] &)[Flatten[Nest[{2 #, 2 # + 1, 2 # + i} &, {0}, n]]]* (compare page 1005)
- *Position[Map[Split, NestList[Sort[Flatten[{#, # + 1}]] &, {0}, 2^n - 1]], _?(OddQ[Length[#]] &), {2}]* (see page 358)
- *Flatten[Table[Map[{t, #} &, Fold[Flatten[{#1, #1 + #2}] &, 0, Flatten[2^(Position[Reverse[IntegerDigits[t, 2]], 1] - 1)]]], {t, 2^n - 1}], 1]* (see page 870)
- *Map[Map[FromDigits[#, 2] &, Transpose[Partition[#, 2]]] &, Position[Nest[{{#, #}, {#}} &, 1, n], 1] - 1]* (see page 509)

A formatting hack giving the same visual pattern is

```
DisplayForm[Nest[SubsuperscriptBox[#, #, #] &, "1", n]]
```

▪ **Non-white backgrounds.** The pictures below show substitution systems in which white squares are replaced by blocks which contain black squares. There is still a nested structure but it is usually not visually as obvious as before. (See page 583.)



▪ **Higher-dimensional generalizations.** The state of a d-dimensional substitution system can be represented by a nested list of depth d. The evolution of the system for t steps can be obtained from

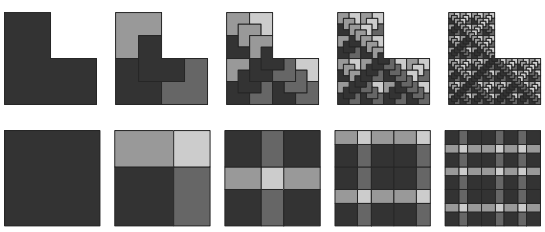
```
SSEvolve[rule_, init_, t_, d_Integer] :=
  Nest[FlattenArray[# /. rule, d] &, init, t]
FlattenArray[list_, d_] :=
  Fold[Function[{a, n}, Map[MapThread[Join, #, n] &,
      a, -{d + 2}]], list, Reverse[Range[d] - 1]]
```

The analog in 3D of the 2D rule on page 187 is

```
{1 → Array[If[LessEqual[##], 0, 1] &, {2, 2, 2}],
  0 → Array[0 &, {2, 2, 2}]}
```

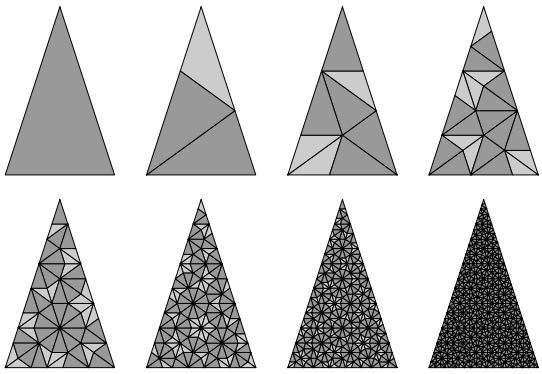
Note that in d dimensions, each black cell must be replaced by at least $d+1$ black cells at each step in order to obtain an object that is not restricted to a dimension $d-1$ hyperplane.

▪ **Other shapes.** The systems on pages 187 and 188 are based on subdividing squares into smaller squares. But one can also set up substitution systems that are based on subdividing other geometrical figures, as shown below.



The second example involves two distinct shapes: a square and a *GoldenRatio* aspect ratio rectangle. Labelling each shape and orientation with a different color, the behavior of this system can be reproduced with equal-sized squares using the rule *{3 → {{1, 0}, {3, 2}}, 2 → {{1}, {3}}, 1 → {{3, 2}}, 0 → {{3}}}* starting from initial condition *{{3}}*.

▪ **Penrose tilings.** The nested pattern shown below was studied by Roger Penrose in 1974 (see page 943).



The arrangement of triangles at step t can be obtained from a substitution system according to

```
With[{ϕ = GoldenRatio}, Nest[# /. a[p_, q_, r_] :>
    With[{s = (p + ϕ q) (2 - ϕ)}, {a[r, s, q], b[r, s, p]}] /.
  b[p_, q_, r_] :> With[{s = (p + ϕ r) (2 - ϕ)}, {a[p, q, s], b[
      r, s, q]}] &, a[{1/2, Sin[2 π/5] ϕ}, {1, 0}, {0, 0}], t]]
```

This pattern can be viewed as generalizations of the pattern generated by the 1D Fibonacci substitution system (c) on page 83. As discussed on page 903, this 1D sequence can be obtained by looking at how a line with *GoldenRatio* slope cuts through a 2D lattice of squares. Penrose tilings can be obtained by looking at how a 2D plane with slopes based on *GoldenRatio* cuts through a lattice of hypercubes in 5D. The tilings turn out to have approximate 5-fold symmetry. (See also page 943.)

In general, projections onto any regular lattice in any number of dimensions from hyperplanes with any quadratic irrational slopes will yield nested patterns that can be generated by subdividing some shape or another according to a substitution system. Despite some confusion in the literature, however, this procedure can reproduce only a tiny fraction of all possible nested patterns.

▪ **Page 189 · Dragon curve.** The pattern shown here can be obtained in several related ways, including from numbers in base $i-1$ (see below) and from a doubled version of the paths generated by 1D paperfolding substitution systems (see page 892). Its boundary has fractal dimension $2\,\text{Log}[2, \text{Root}[2 + \#1^2 - \#1^3, 1]] \approx 1.52$.

▪ **Implementation.** The most convenient approach is to represent each pattern by a list of complex numbers, with the center of each square being given in terms of each complex number *z* by *{Re[z], Im[z]}*. The pattern after *n* steps is then given by *Nest[Flatten[f[#]] &, {0}, n]*, where for the rule on page 189 *f[z_] = 1/2 (1 - i) {z + 1/2, z - 1/2}* (*f[z_] = (1 - i) {z + 1, z}* gives a transformed version). For the rule on page 190, *f[z_] = 1/2 (1 - i) {i z + 1/2, z - 1/2}*. For rules (a), (b) and (c) (Koch curve) on page 191 the forms of *f[z_]* are respectively:

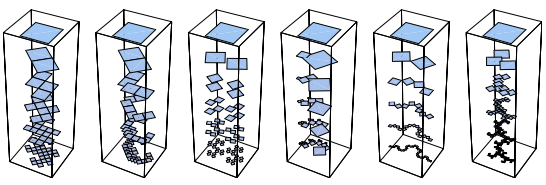
```
(0.296 - 0.57 i) z - 0.067 i - {1.04, 0.237}
N[1/40 {17 (√3 - i) z, -24 + 14 z}]
N[((1/2 (1/√3 - 1) (i + {1, -1}) - i - (1 + {i, -i}/√3) z)/2]
```

▪ **Connection with digit sequences.** Patterns after *t* steps can be viewed as containing all *t*-digit integers in an appropriate complex base. Thus the patterns on page 189 can be formed from *t*-digit integers in base *i - 1* containing only digits 0 and 1, as given by

```
Table[FromDigits[IntegerDigits[s, 2, t], i - 1], {s, 0, 2^t - 1}]
```

In the particular case of base *i - q* with digits 0 through q^2, it turns out that for sufficiently large *t* any complex integer can be represented, and will therefore be part of the pattern. (Compare page 1094.)

▪ **Visualization.** The 3D pictures below show successive steps in the evolution of each of the geometric substitution systems from the main text.

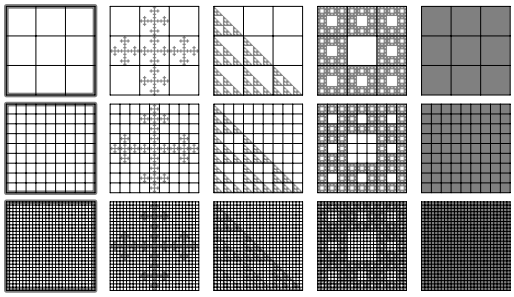


▪ **Parameter space sets.** See pages 407 and 1006 for a discussion of varying parameters in geometrical substitution systems.

▪ **Affine transformations.** Any set of so-called affine transformations that take the vector for each point, multiply it by a fixed matrix and then add a fixed vector, will yield nested patterns similar to those shown in the main text. Linear operations on complex numbers of the kind discussed above correspond geometrically to rotations, translations and rescalings. General affine transformations also allow reflection and skewing. In addition, affine transformations can readily be generalized to any number of dimensions, while complex numbers represent only two dimensions.

▪ **Complex maps.** Many kinds of nonlinear transformations on complex numbers yield nested patterns. Sets of so-called Möbius transformations of the form *z → (a z + b)/(c z + d)* always yield such patterns (and correspond to so-called modular groups when *a d - b c == 1*). Transformations of the form *z → {Sqrt[z - c], -Sqrt[z - c]}* yield so-called Julia sets which form nested patterns for many values of *c* (see note below). In fact, a fair fraction of all possible transformations based on algebraic functions will yield nested patterns. For typically the continuity of such functions implies that only a limited number of shapes not related by limited variations in local magnification can occur at any scale.

▪ **Fractal dimensions.** Certain features of nested patterns can be characterized by so-called fractal dimensions. The pictures below show five patterns with three successively finer grids superimposed. The dimension of a pattern can be computed by looking at how the number of grid squares that have any gray in them varies with the length *a* of the edge of each grid square. In the first case shown, this number varies like $(1/a)^1$ for small *a*, while in the last case, it varies like $(1/a)^2$. In general, if the number varies like $(1/a)^d$, one can take *d* to be the dimension of the pattern. And in the intermediate cases shown, it turns out that *d* has non-integer values.

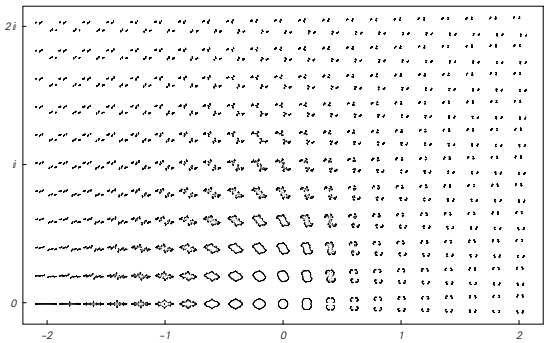


The grid in the pictures above fits over the pattern in a very regular way. But even when this does not happen, the limiting behavior for small *a* is still $(1/a)^d$ for any nested pattern. This form is inevitable if the underlying pattern effectively has the same structure on all scales. For some of the more complex patterns encountered in this book, however, there continues to be different structure on different scales, so that the effective value of *d* fluctuates as the scale changes, and may not converge to any definite value. (Precise definitions of dimension based for example on the maximum ever achieved by *d* will often in general imply formally non-computable values, as in the discussion of page 1138.)

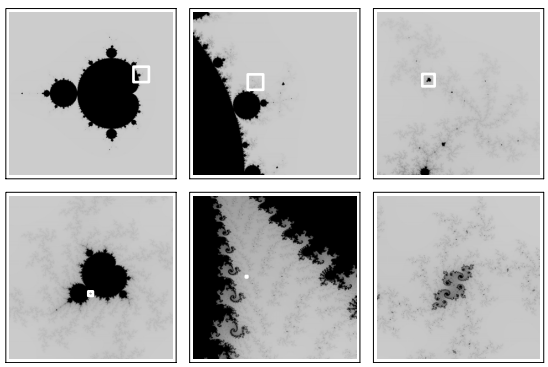
Fractal dimensions characterize some aspects of nested patterns, but patterns with the same dimension can often look very different. One approach to getting better characterizations is to look at each grid square, and to ask not just whether there is any gray in it, but how much. Quantities derived from the mean, variance and other moments of the probability distribution can serve as generalizations of fractal dimension. (Compare page 959.)

■ **History of fractals.** The idea of using nested 2D shapes in art probably goes back to antiquity; some examples were shown on page 43. In mathematics, nested shapes began to be used at the end of the 1800s, mainly as counterexamples to ideas about continuity that had grown out of work on calculus. The first examples were graphs of functions: the curve on page 918 was discussed by Bernhard Riemann in 1861 and by Karl Weierstrass in 1872. Later came geometrical figures: example (c) on page 191 was introduced by Helge von Koch in 1906, the example on page 187 by Wacław Sierpiński in 1916, examples (a) and (c) on page 188 by Karl Menger in 1926 and the example on page 190 by Paul Lévy in 1937. Similar figures were also produced independently in the 1960s in the course of early experiments with computer graphics, primarily at MIT. From the point of view of mathematics, however, nested shapes tended to be viewed as rare and pathological examples, of no general significance. But the crucial idea that was developed by Benoit Mandelbrot in the late 1960s and early 1970s was that in fact nested shapes can be identified in a great many natural systems and in several branches of mathematics. Using early raster-based computer display technology, Mandelbrot was able to produce striking pictures of what he called fractals. And following the publication of Mandelbrot's 1975 book, interest in fractals increased rapidly. Quantitative comparisons of pure power laws implied by the simplest fractals with observations of natural systems have had somewhat mixed success, leading to the introduction of multifractals with more parameters, but Mandelbrot's general idea of the importance of fractals is now well established in both science and mathematics.

■ **The Mandelbrot set.** The pictures below show Julia sets produced by the procedure of taking the transformation $z \to \{Sqrt[z - c], -Sqrt[z - c]\}$ discussed above and iterating it starting at $z = 0$ for an array of values of c in the complex plane.

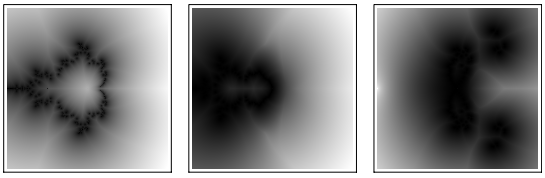


The Mandelbrot set introduced by Benoit Mandelbrot in 1979 is defined as the set of values of c for which such Julia sets are connected. This turns out to be equivalent to the set of values of c for which starting at $z = 0$ the inverse mapping $z \to z^2 + c$ leads only to bounded values of z. The Mandelbrot set turns out to have many intricate features which have been widely reproduced for their aesthetic value, as well as studied by mathematicians. The first picture below shows the overall form of the set; subsequent pictures show successive magnifications of the regions indicated. All parts of the Mandelbrot set are known to be connected. The whole set is not self-similar. However, as seen in the third and fourth pictures, within the set are isolated small copies of the whole set. In addition, as seen in the last picture, near most values of c the boundary of the Mandelbrot set looks very much like the Julia set for that value of c.



On pages 407 and 1006 I discuss parameter space sets that are somewhat analogous to the Mandelbrot set, but whose properties are in many respects much clearer. And from this discussion there emerges the following interpretation of the Mandelbrot set that appears not to be well known but which I find most illuminating. Look at the array of Julia sets and ask for each c whether the Julia set includes the point $z = 0$.

The set of values of c for which it does corresponds exactly to the boundary of the Mandelbrot set. The pictures below show a generalization of this idea, in which gray level indicates the minimum distance *Abs[z - z0]* of any point z in the Julia set from a fixed point z_0. The first picture shows the case $z_0 = 0$, corresponding to the usual Mandelbrot set.



$z_0 = 0$ $z_0 = 1$ $z_0 = i$

▪ **Page 192 · Neighbor-dependent substitution systems.** Given a list of individual replacement rules such as *{{_, 1}, {0, 1}} → {{1, 0}, {1, 1}}*, each step in the evolution shown corresponds to

```
Flatten2D[Partition[list, {2, 2}, 1, -1] /. rule]
```

One can consider rules in which some replacements lead to subdivision of elements but others do not. However, unlike for the 1D case, there will in general in 2D be an arbitrarily large set of different possible neighborhood configurations around any given cell.

▪ **Page 192 · Space-filling curves.** One can conveniently scan a finite 2D grid just by going along each successive row in turn. One can scan a quadrant of an infinite grid using the σ function on page 1127, or one can scan a whole grid by for example going in a square spiral that at step t reaches position

```
(1/2 (-1)^# ({1, -1} (Abs[#^2 - t] - #) + #^2 - t - Mod[#, 2]) &)[
  Round[√t ]]
```

Network Systems

▪ **Implementation.** The nodes in a network system can conveniently be labelled by numbers *1*, *2*, ... *n*, and the network obtained at a particular step can be represented by a list of pairs, where the pair at position i gives the numbers corresponding to the nodes reached by following the above and below connections from node i. With this setup, a network consisting of just one node is *{{1, 1}}* and a 1D array of n nodes can be obtained with

```
CyclicNet[n_] := RotateRight[
    Table[Mod[{i - 1, i + 1}, n] + 1, {i, n}]]
```

With above connections represented as *1* and the below connections as *2*, the node reached by following a succession s of connections from node i is given by

```
Follow[list_, i_, s_List] := Fold[list[[#1]][[#2]] &, i, s]
```

The total number of distinct nodes reached by following all possible succession of connections up to length d is given by

```
NeighborNumbers[list_, i_Integer, d_Integer] :=
  Map[Length, NestList[Union[Flatten[list[[#]]]] &,
      Union[list[[i]]], d - 1]]
```

For each such list the rules for the network system then specify how the connections from node i should be rerouted. The rule *{2, 3} → {{2, 1}, {1}}* specifies that when *NeighborNumbers* gives *{2, 3}* for a node i, the connections from that node should become *{Follow[list, i, {2, 1}], Follow[list, i, {1}]}*. The rule *{2, 3} → {{{2, 1}, {1, 1}}, {1}}* specifies that a new node should be inserted in the above connection, and this new node should have connections *{Follow[list, i, {2, 1}], Follow[list, i, {1, 1}]}*. With rules set up in this way, each step in the evolution of a network system is given by

```
NetEvolveStep[{depth_Integer, rule_List}, list_List] := Block[
  {new = {}}, Join[Table[Map[NetEvolveStep1[#, list, i] &,
      Replace[NeighborNumbers[list, i, depth],
       rule]], {i, Length[list]}], new]]

NetEvolveStep1[s : {___Integer}, list_, i_] := Follow[list, i, s]

NetEvolveStep1[{s1 : {___Integer}, s2 : {___Integer}},
    list_, i_] := Length[list] + Length[
     AppendTo[new, {Follow[list, i, s1], Follow[list, i, s2]}]]
```

The set of nodes that can be reached from node i is given by

```
ConnectedNodes[list_, i_] :=
  FixedPoint[Union[Flatten[{#, list[[#]]}]] &, {i}]
```

and disconnected nodes can be removed using

```
RenumberNodes[list_, seq_] :=
  Map[Position[seq, #][[1, 1]] &, list[[seq]], {2}]
```

The sequence of networks obtained on successive steps by applying the rules and then removing all nodes not connected to node number *1* is given by

```
NetEvolveList[rule_, init_, t_Integer] :=
  NestList[(RenumberNodes[#, ConnectedNodes[#, 1]] &)[
      NetEvolveStep[rule, #]] &, init, t]
```

Note that the nodes in each network are not necessarily numbered in the order that they appear on successive lines in the pictures in the main text. Additional information on the origin of each new node must be maintained if this order is to be found.

▪ **Rule structure.** For depth 1, the possible results from *NeighborNumbers* are *{1}* and *{2}*. For depth 2, they are *{1, 1}*, *{1, 2}*, *{2, 1}*, *{2, 2}*, *{2, 3}* and *{2, 4}*. In general, each successive element in a list from *NeighborNumbers* cannot be more than twice the previous element.

▪ **Undirected networks.** Networks with connections that do not have definite directions are discussed at length in Chapter 9, mainly as potential models for space in the universe. The rules for updating such networks turn out to be

somewhat more difficult to apply than those for the network systems discussed here.

■ **Page 199 · Computer science.** The networks discussed here can be thought of as very simple analogs of data structures in practical computer programs. The connections correspond to pointers between elements of these data structures. The fact that there are two connections coming from each node is reminiscent of the LISP language, but in the networks considered here there are no leaves corresponding to atoms in LISP. Note that the process of dropping nodes that become disconnected is analogous to so-called "garbage collection" for data structures. The networks considered here are also related to the combinator systems discussed on page 1121.

■ **Page 202 · Properties.** Random behavior seems to occur in a few out of every thousand randomly selected rules of the kind shown here. In case (c), the following gives a list of the numbers of nodes generated up to step t:

```
FoldList[Plus, 1, Join[{1, 4, 12, 10, -20, 6, 4},
    Map[d, IntegerDigits[Range[4, t - 5], 2]]]]
```

```
d[{___, 1}] = 1
```

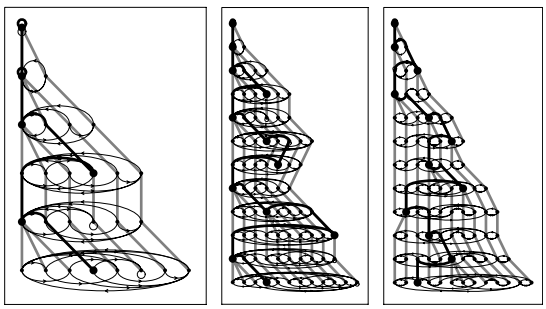
```
d[{1, p : ((0) ..), 0}] :=
  -Apply[Plus, 4 Range[Length[{p}]] - 1] + 6
```

```
d[{__, 1, p : ((0) ..), 0}] := d[{1, p, 0}] - 7
```

```
d[{___, p : ((1) ..), q : ((0) ...), 1, 0}] :=
  4 Length[{p}] + 3 Length[{q}] + 2
```

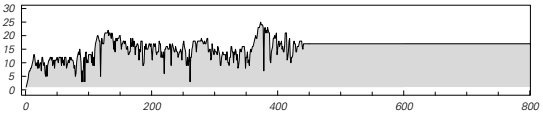
```
d[{___, p : ((1) ..), 1, 0}] := 4 Length[{p}] + 2
```

■ **Sequential network systems.** In the network systems discussed in the main text, every node is updated in parallel at each step. It is however also possible to consider systems in which there is only a single active node, and operations are performed only on that node at any particular step. The active node can move by following its above or below connections, in a way that is determined by a rule which depends on the local structure of the network. The pictures below show examples of sequential network systems; the path of the active node is indicated by a thick black line.

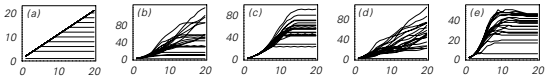


It is rather common for the active node eventually to get stuck at a particular position in the network; the picture below shows the effect of this on the total number of nodes in the last case illustrated above. The rule for this system is

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



■ **Dimensionality of networks.** As discussed on page 479, if a sufficiently large network has a d-dimensional form, then by following r connections in succession from a given node, one should reach about r^d distinct nodes. The plots below show the actual numbers of nodes reached as a function of r for the systems on pages 202 and 203 at steps 1, 10, 20, ..., 200.



■ **Cellular automata on networks.** The cellular automata that we have considered so far all have cells arranged in regular arrays. But one can also set up generalizations in which the cells correspond to nodes in arbitrary networks. Given a network of the kind discussed in the main text of this section, one can assign a color to each node, and then update this color at each step according to a rule that depends on the colors of the nodes to which the connections from that node go. The behavior obtained depends greatly on the form of the network, but with networks of finite size the results are typically like those obtained for other finite size cellular automata of the kind discussed on page 259.

■ **Implementation.** Given a network represented as a list in which element i is $\{a, i, b\}$, where a is the node reached by the above connection from node i, and b is the node reached by the below connection, each step corresponds to

```
NetCAStep[{rule_, net_}, list_] :=
  Map[Replace[#, rule] &, list[[net]]]
```

■ **Boolean networks.** Several lines of development from the cybernetics movement (notably in immunology, genetics and management science) led in the 1960s to a study of random Boolean networks—notably by Stuart Kauffman and Crayton Walker. Such systems are like cellular automata on networks, except for the fact that when they are set up each node has a rule that is randomly chosen from all 2^{2^s} possible ones with s inputs. With $s = 2$ class 2 behavior (see Chapter 6) tends to

dominate. But for $s > 2$, the behavior one sees quickly approaches what is typical for a random mapping in which the network representing the evolution of the 2^m states of the m underlying nodes is itself connected essentially randomly (see page 963). (Attempts were made in the 1980s to study phase transitions as a function of s in analogy to ones in percolation and spin glasses.) Note that in almost all work on random Boolean networks averages are in effect taken over possible configurations, making it impossible to see anything like the kind of complex behavior that I discuss in cellular automata and many other systems in this book.

Multiway Systems

▪ **Implementation.** It is convenient to represent the state of a multiway system at each step by a list of strings, where an individual string is for example *"ABBAAB"*. The rules for the multiway system can then be given for example as

```
{"AAB" → " BB", " BA" → " ABB"}
```

The evolution of the system is given by the functions

```
MWStep[rule_List, slist_List] := Union[Flatten[
      Map[Function[s, Map[MWStep1[#, s] &, rule]], slist]]]
MWStep1[p_String → q_String, s_String] :=
  Map[StringReplacePart[s, q, #] &, StringPosition[s, p]]
MWEvolveList[rule_, init_List, t_Integer] :=
  NestList[MWStep[rule, #] &, init, t]
```

An alternative approach uses lists instead of strings, and in effect works by tracing the internal steps that *Mathematica* goes through in trying out possible matchings. With the rule from above written as

```
{{x___, 0, 0, 1, y___} → {x, 1, 1, y},
  {x___, 1, 0, y___} → {x, 0, 1, 1, y}}
```

MWStep can be rewritten as

```
MWStep[rule_List, slist_List] :=
  Union[Flatten[Map[ReplaceList[#, rule] &, slist], 1]]
```

The case shown on page 206 is

```
{"AB" → "", "ABA" → "ABBAB", "ABABBB" → "AAAAABA"}
```

starting with *{"ABABAB"}*. Note that the rules are set up so that a string for which there are no applicable replacements at a given step is simply dropped.

▪ **General properties.** The merging of states (as done above by *Union*) is crucial to the behavior seen. Note that the pictures shown indicate only which states yield which states—not for example in how many ways the rules can be applied to a given state to yield a given new state.

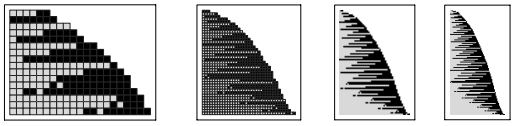
If there was no merging, then if a typical state yielded more than one new state, then inevitably the total number of states would increase exponentially. But when there is merging, this need not occur—making it difficult to give probabilistic estimates of growth rates. Note that a given rule can yield very different growth rates with different initial conditions. Thus, for example, the growth rate for *{"A" → "AA", "AB" → "BA", "BA" → "AB"}* is t^{n+1}, where n is the number of initial B's. With most rules, states that appear at one step can disappear at later steps. But if *"A" → "A"* and its analogs are part of the rule, then every state will always be kept, almost inevitably leading to overall nesting in pictures like those on page 208.

In cases where all strings that appear both in rules and initial conditions are sorted—so that for example A's appear before B's—any string generated will also be sorted, so it can be specified just by giving a list of how many A's and how many B's appear in it. The rule for the system can then be stated in terms of a difference vector—which for *{"BA" → "AAA", "BAA" → "BBBA"}* is *{{2, -1}, {-1, 2}}*. Given a list of string specifications, a step in the evolution of the multiway system corresponds to

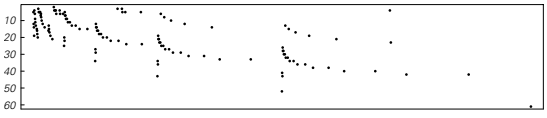
```
Select[Union[Flatten[Outer[Plus, diff, list, 1], 1]],
  Abs[#] == # &]
```

▪ **Page 206 · Properties.** The total number of strings grows approximately quadratically; its differences repeat (offset by 1) with period 1071. The number of new strings generated at successive steps grows approximately linearly; its differences repeat with period 21. The third element of the rule is at first used only on some steps—but after step 50 it appears to be used somewhere in every step.

The pictures below show in stacked form (as on page 208) all sequences generated at various steps of evolution. Note that after just a few steps, the sequences produced always seem to consist of white elements followed by black, with possibly one block of black in the white region. Without this additional block of black, only the first case in the rule can ever apply.



In analogy with page 796 the picture below shows when different strings with lengths up to 10 are reached in the evolution of the system.



Different initial conditions for this multiway system lead to behavior that either dies out (as for *"ABA"*), or grows exponentially forever (as for *"ABAABABA"*).

■ **Frequency of behavior.** Among multiway systems with randomly chosen rules, one finds about equal numbers that grow rapidly and die out completely. A few percent exhibit repetitive behavior, while only one in several million exhibit more complex behavior. One common form of more complex behavior is quadratic growth, with essentially periodic fluctuations superimposed—as on page 206.

■ **History.** Versions of multiway systems have been invented many times in a variety of contexts. In mathematics specific examples of them arose in formal group theory (see below) around the end of the 1800s. Axel Thue considered versions with two-way rules (analogous to semigroups, as discussed below) in 1912, leading to the name semi-Thue systems sometimes being used for general multiway systems. Other names for multiway systems have included string and term rewrite systems, production systems and associative calculi. From the early 1900s various generalizations of multiway systems were used as idealizations of mathematical proofs (see page 1150); multiway systems with explicit pattern variables (such as *s_*) were studied under the name canonical systems by Emil Post starting in the 1920s. Since the 1950s, multiway systems have been widely used as generators of formal languages (see below). Simple analogs of multiway systems have also been used in genetic analysis in biology and in models for particle showers and other branching processes in physics and elsewhere.

■ **Semigroups and groups.** The multiway systems that I discuss can be viewed as representations for generalized versions of familiar mathematical structures. Semigroups are obtained by requiring that rules come in pairs: with each rule such as *"ABB"* → *"BA"* there must also be the reversed rule *"BA"* → *"ABB"*. Such pairs of rules correspond to relations in the semigroup, specifying for example that *"ABB"* is equivalent to *"BA"*. (The operation in the semigroup is concatenation of strings; *""* acts as an identity element, so in fact a monoid is always obtained.) Groups require that not only rules but also symbols come in pairs. Thus, for example, in addition to a symbol *A*, there must be an inverse symbol *a*, with the rules *"Aa"* → *""*, *"aA"* → *""* and their reversals.

In the usual mathematical approach, the objects of greatest interest for many purposes are those collections of sequences that cannot be transformed into each other by any of the rules given. Such collections correspond to distinct elements of the group or semigroup, and in general many different choices of underlying rules may yield the same elements with the same properties. In terms of multiway systems, each of the elements corresponds to a disconnected part of the network formed from all possible sequences.

Given a particular representation of a group or semigroup in terms of rules for a multiway system, an object that is often useful is the so-called Cayley graph—a network where each node is an element of the group, and the connections show what elements are reached by appending each possible symbol to the sequences that represent a given element. The so-called free semigroup has no relations and thus no rules, so that all strings of generators correspond to distinct elements, and the Cayley graph is a tree like the ones shown on page 196. The simplest non-trivial commutative semigroup has rules *"AB"* → *"BA"* and *"BA"* → *"AB"*, so that strings of generators with *A*'s and *B*'s in different orders are equivalent and the Cayley graph is a 2D grid.

For some sets of underlying rules, the total number of distinct elements in a group or semigroup is finite. (Compare page 945.) A major mathematical achievement in the 1980s was the complete classification of all possible so-called simple finite groups that in effect have no factors. (For semigroups no such classification has yet been made.) In each case, there are many different choices of rules that yield the same group (and similar Cayley graphs). And it is known that even fairly simple sets of rules can yield large and complicated groups. The icosahedral group A_5 defined by the rules $x^2 == y^3 == (x\,y)^5 == 1$ has 60 elements. But in the most complicated case a dozen rules yield the Monster Group, where the number of elements is

808017424794512875886459904961710757005754368000000000

(See also pages 945 and 1032.)

Following work in the 1980s and 1990s by Mikhael Gromov and others, it is also known that for groups with randomly chosen underlying rules, the Cayley graph is usually either finite, or has a rapidly branching tree-like structure. But there are presumably also marginal cases that exhibit complex behavior analogous to what we saw in the main text. And indeed for example, despite conjectures to the contrary, it was found in the 1980s by Rostislav Grigorchuk that complicated groups could be constructed in which growth intermediate between polynomial and exponential can occur. (Note that different choices of generators can yield Cayley graphs with different local subgraphs; but the overall structure of a sufficiently large graph for a particular group is always the same.)

■ **Formal languages.** The multiway systems that I discuss are similar to so-called generative grammars in the theory of formal languages. The idea of a generative grammar is that

all possible expressions in a particular formal language can be produced by applying in all possible ways the set of replacement rules given by the grammar. Thus, for example, the rules `{"x" → "xx", "x" → "(x)", "x" → "()"}` starting with `"x"` will generate all expressions that consist of balanced sequences of parentheses. (Final expressions correspond to those without the "non-terminal" symbol *x*.) The hierarchy described by Noam Chomsky in 1956 distinguishes four kinds of generative grammars (see page 1104):

Regular grammars. The left-hand side of each rule must consist of one non-terminal symbol, and the right-hand side can contain only one non-terminal symbol. An example is `{"x" → "xA", "x" → "yB", "y" → "xA"}` starting with `"x"` which generates sequences in which no pair of *B*'s ever appear together. Expressions in regular languages can be recognized by finite automata of the kind discussed on page 957.

Context-free grammars. The left-hand side of each rule must consist of one non-terminal symbol, but the right-hand side can contain several non-terminal symbols. Examples include the parenthesis language mentioned above, `{"x" → "AxA", "x" → "B"}` starting with `"x"`, and the syntactic definitions of *Mathematica* and most other modern computer languages. Context-free languages can be recognized by a computer using only memory on a single last-in first-out stack. (See pages 1091 and 1103.)

Context-sensitive grammars. The left-hand side of each rule is no longer than the right, but is otherwise unrestricted. An example is `{"Ax" → "AAxx", "xA" → "BAA", "xB" → "Bx"}` starting with `"AAxBA"`, which generates expressions of the form `Table["A", {n}] <> Table["B", {n}] <> Table["A", {n}]`.

Unrestricted grammars. Any rules are allowed.

(See also page 944.)

▪ **Multidimensional multiway systems.** As a generalization of multiway systems based on 1D strings one can consider systems in which rules operate on arbitrary blocks of elements in an array in any number of dimensions. Still more general network substitution systems are discussed on page 508.

▪ **Limited size versions.** One can set up multiway systems of limited size by applying transformations cyclically to strings.

▪ **Multiway tag systems.** See page 1141.

▪ **Multiway systems based on numbers.** One can consider for example the rule `n → {n + 1, 2 n}` implemented by

```
NestList[Union[Flatten[{# + 1, 2 #}]] &, {0}, t]
```

In this case there are `Fibonacci[t + 2]` distinct numbers obtained at step *t*. In general, rules based on simple arithmetic operations yield only simple nested structures. If the numbers *n* are allowed to have both real and imaginary parts then results analogous to those discussed for substitution systems on page 933 are obtained. (Somewhat related systems based on recursive sequences are discussed on page 907. Compare also sorted multiway systems on page 937.)

▪ **Non-deterministic systems.** Multiway systems are examples of what are often in computer science called non-deterministic systems. The general idea of a non-deterministic system is to have rules with several possible outcomes, and then to allow each of these outcomes to be followed. Non-deterministic Turing machines are a common example. For most types of systems (such as Turing machines) such non-deterministic versions do not ultimately allow any greater range of computations to be performed than deterministic ones. (But see page 766.)

▪ **Fundamental physics.** See page 504.

▪ **Game systems.** One can think of positions or configurations in a game as corresponding to nodes in a large network, and the possible moves in the game as corresponding to connections between nodes. Most games have rules which imply that if certain states are reached one player can be forced in the end to lose, regardless of what specific moves they make. And even though the underlying rules in the game may be simple, the pattern of such winning positions is often quite complex. Most games have huge networks whose structure is difficult to visualize (even the network for tic-tac-toe, for example, has 5478 nodes). One example that allows easy visualization is a simplification of several common games known as nim. This has *k* piles of objects, and on alternate steps each of two players takes as many objects as they want from any one of the piles. The winner is the player who manages to take the very last object. With just two piles one player can force the other to lose by arranging that after each of their moves the two piles have equal heights. With more than two piles it was discovered in 1901 that one player can in general force the other to lose by arranging that after each of their moves `Apply[BitXor, h] == 0`, where *h* is the list of heights. For `k > 1` this yields a nested pattern, analogous to those shown on page 871. If one allows only specific numbers of objects to be taken at each step a nested pattern is again obtained. With more general rules it seems almost inevitable that much more complicated patterns will occur.

Systems Based on Constraints

■ **The notion of equations.** In the mathematical framework traditionally used in the exact sciences, laws of nature are usually represented not by explicit rules for evolution, but rather by abstract equations. And in general what such equations do is to specify constraints that systems must satisfy. Sometimes these constraints just relate the state of a system at one time to its state at a previous time. And in such cases, the constraints can usually be converted into explicit evolution rules. But if the constraints relate different features of a system at one particular time, then they cannot be converted into evolution rules. In computer programs and other kinds of discrete systems, explicit evolution rules and implicit constraints usually work very differently. But in traditional continuous mathematics, it turns out that these differences are somewhat obscured. First of all, at a formal level, equations corresponding to these two cases can look very similar. And secondly, the equations are almost always so difficult to deal with at all that distinctions between the two cases are not readily noticed.

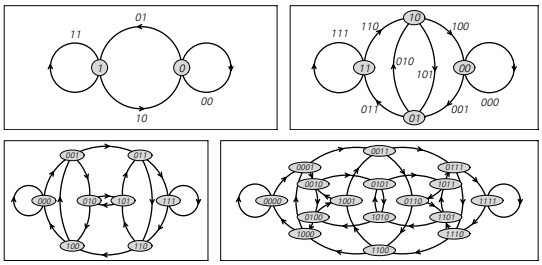
In the language of differential equations—the most widely used models in traditional science—the two cases we are discussing are essentially so-called initial value and boundary value problems, discussed on page 923. And at a formal level, the two cases are so similar that in studying partial differential equations one often starts with an equation, and only later tries to work out whether initial or boundary values are needed in order to get either any solution or a unique solution. For the specific case of second-order equations, it is known in general what is needed. Elliptic equations such as the Laplace equation need boundary values, while hyperbolic and parabolic equations such as the wave equation and diffusion equation need initial values. But for higher-order equations it can be extremely difficult to work out what initial or boundary values are needed, and indeed this has been the subject of much research for many decades.

Given a partial differential equation with initial or boundary values, there is then the question of solving it. To do this on a computer requires constructing a discrete approximation. But it turns out that the standard methods used (such as finite difference and finite element) involve extremely similar computations for initial and for boundary value problems, leaving no trace of the significant differences between these cases that are so obvious in the discrete systems that we discuss in most of this book.

■ **Linear and nonlinear systems.** A vast number of different applications of traditional mathematics are ultimately based on linear equations of the form $u == m \,.\, v$ where u and v are vectors (lists) and m is a matrix (list of lists), all containing ordinary continuous numbers. If v is known then such equations in essence provide explicit rules for computing u. But if only u is known, then the equations can instead be thought of as providing implicit constraints for v. However, it so happens that even in this case v can still be found fairly straightforwardly using *LinearSolve[m, u]*. With vectors of length n it generically takes about n^2 steps to compute u given v, and a little less than n^3 steps to compute v given u (the best known algorithms—which are based on matrix multiplication—currently involve about $n^{2.4}$ steps). But as soon as the original equation is nonlinear, say $u == m_1 \,.\, v + m_2 \,.\, v^2$, the situation changes dramatically. It still takes only about n^2 steps to compute u given v, but it becomes vastly more difficult to compute v given u, taking perhaps 2^{2^n} steps. (Generically there are 2^n solutions for v, and even for integer coefficients in the range $-r$ to $+r$ already in 95% of cases there are 4 solutions with $n = 2$ as soon as $r \geq 6$.)

■ **Explanations based on constraints.** In some areas of science it is common to give explanations in terms of constraints rather than mechanisms. Thus, for example, in physics there are so-called variational principles which state that physical systems will behave in ways that minimize or maximize certain quantities. One such principle implies that atoms in molecules will tend to arrange themselves so as to minimize their energy. For simple molecules, this is a useful principle. But for complicated molecules of the kind that are common in living systems, this principle becomes much less useful. In fact, in finding out what configuration such molecules actually adopt, it is usually much more relevant to know how the molecule evolves in time as it is created than which of its configurations formally has minimum energy. (See pages 342 and 1185.)

■ **Page 211 · 1D constraints.** The constraints in the main text can be thought of as specifying that only some of the k^n possible blocks of cells of length n (with k possible colors for each cell) are allowed. To see the consequences of such constraints consider breaking a sequence of colors into blocks of length n, with each block overlapping by $n-1$ cells with its predecessor, as in *Partition[list, n, 1]*. If all possible sequences of colors were allowed, then there would be k possibilities for what block could follow a given block, given by *Map[Rest, Table[Append[list, i], {i, 0, k - 1}]]*. The possible sequences of length n blocks that can occur are conveniently represented by possible paths by so-called de Bruijn networks, of the kind shown for $k = 2$ and $n = 2$ through 5 below.



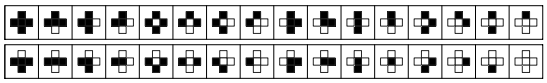
Given the network for a particular n, it is straightforward to see what happens when only certain length n blocks are allowed: one just keeps the arcs in the network that correspond to allowed blocks, and drops all other ones. Then if one can still form an infinite path by going along the arcs that remain, this path will correspond to a pattern that satisfies the constraints. Sometimes there will be a unique such path; in other cases there will be choices that can be made along the path. But the crucial point is that since there are only k^{n-1} nodes in the network, then if any infinite path is possible, there must be such a path that visits the same node and thus repeats itself after at most k^{n-1} cells. The constraint on page 210 has $k = 2$ and $n = 3$; the pattern that satisfies it repeats with period 4, thus saturating the bound. (See also page 266.)

▪ **1D cellular automata.** In a cellular automaton with k colors and r neighbors, configurations that are left invariant after t steps of evolution according to the cellular automaton rule are exactly the ones which contain only those length $2r + 1$ blocks in which the center cell is the same before and after the evolution. Such configurations therefore obey constraints of the kind discussed in the main text. As we will see on page 225 some cellular automata evolve to invariant configurations from any initial conditions, but most do not. (See page 954.)

▪ **Dynamical systems theory.** Sets of sequences in which a finite collection of blocks are excluded are sometimes known as finite complement languages, or subshifts of finite type. (See page 958.)

▪ **Page 215 · 2D constraints.** The constraints shown here are minimal, in the sense that in each case removing any of the allowed templates prevents the constraint from ever being satisfied. Note that constraints which differ only by overall rotation, reflection or interchange of black and white are not explicitly shown. The number of allowed templates out of the total of 32 possible varies from 1 to 15 for the constraints shown, with 12 being the most common. Smaller sets of allowed templates typically seem to lead to constraints that can be satisfied by visually simpler patterns.

▪ **Numbering scheme.** The constraint numbered n allows the templates at *Position[IntegerDigits[n, 2, 32], 1]* in the list below. (See also page 927.)



▪ **Identifying the 171 patterns.** The number of constraints to consider can be reduced by symmetries, by discarding sets of templates that are supersets of ones already known to be satisfiable, and by requiring that each template in the set be compatible with itself or with at least one other in each of the eight immediately adjacent positions. The remaining constraints can then be analyzed by attempting to build up explicit patterns that satisfy them, as discussed below.

▪ **Checking constraints.** A set of allowed templates can be specified by a *Mathematica* pattern of the form $t_1 | t_2 | t_3$ etc. where the t_i are for example *{{_, 1, _}, {0, 0, 1}, {_, 0, _}}*. To check whether an array *list* contains only arrangements of colors corresponding to allowed templates one can then use

```
SatisfiedQ[list_, allowed_] :=
  Apply[And, Map[MatchQ[#, allowed] &,
      Partition[list, {3, 3}, {1, 1}], {2}], {0, 1}]
```

▪ **Representing repetitive patterns.** Repetitive patterns are often most conveniently represented as tessellations of rectangles whose corners overlap. Pattern (a) on page 213 can be specified as

```
{{2, -1, 2, 3}, {{0, 0, 0, 0}, {1, 1, 0, 0}, {1, 0, 0, 0}}}
```

Given this, a complete nx by ny array filled with this pattern can be constructed from

```
c[{d1_, d2_, d3_, d4_}, {x_, y_}] :=
  With[{d = d1 d2 + d1 d4 + d3 d4},
    Mod[{{d2 x + d4 x + d3 y, d4 x - d1 y}}/d, 1]]
Fill[{dlist_, data_}, {nx_, ny_}] :=
 Array[c[dlist, {##}] &, {nx, ny}] /. Flatten[MapIndexed[
    c[dlist, Reverse[#2]] → #1 &, Reverse[data], {2}], 1]
```

▪ **Searching for patterns.** The basic approach to finding a pattern which satisfies a particular constraint on an infinite array of cells is to start with a pattern which satisfies the constraint in a small region, and then to try to extend the pattern. Often the constraint will immediately force a unique extension of the pattern, at least for some distance. But eventually there will normally be places where the pattern is not yet uniquely determined, and so a series of choices have to be made. The procedure used to find the results in this book attempts to extend patterns along a square spiral, making whatever choices are needed, and backtracking if these turn out to be inconsistent with the constraint. At every step in the procedure, regularities are tested for that would imply the possibility of an infinite repetitive pattern. In

addition, whenever there is a choice, the first cases to be tried are set up to be ones that tend to extend whatever regularity has developed so far. And when backtracking is needed, the procedure always goes back to the most recent choice that actually affected whatever inconsistency was discovered. And in addition it remembers what has already been worked out, so as to avoid, for example, unnecessarily working out the pattern on the opposite side of the spiral again.

■ **Undecidability.** The general problem of whether an infinite pattern exists that satisfies a particular constraint is formally undecidable (see page 1139). This means that in general there can be no upper bound on the size of region for which the constraints can be satisfied, even if they are not satisfiable for the complete infinite grid.

■ **NP completeness.** The problem of whether a pattern can be found that satisfies a constraint even in a finite region is NP-complete. (See page 1145.) This suggests that to determine whether a repetitive pattern with repeating blocks of size n exists may in general take a number of steps which grows more rapidly than any polynomial in n.

■ **Enumerating patterns.** Compare page 959.

■ **Page 219 · Non-periodic pattern.** The color at position x,y in the pattern is given by

```
a[x_, y_] := Mod[y + 1, 2] /; x + y > 0
a[x_, y_] := 0 /; Mod[x + y, 2] == 1
a[x_, y_] :=
  Mod[Floor[(x - y) 2^((x+y-6)/4)], 2] /; Mod[x + y, 4] == 2
a[x_, y_] := 1 - Sign[Mod[x - y + 2, 2^((-x-y+8)/4)]]
```

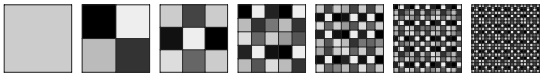
The origin of the x,y coordinates is the only freedom in this pattern. The nested structure is like the progression of base 2 digit sequences shown on page 117. Negative numbers are effectively represented by complements of digit sequences, much as in typical practical computers. With the procedure described above for finding patterns that satisfy a constraint, generating the pattern shown here is straightforward once the appropriate constraint is identified.

■ **Other types of constraints.** Constraints based on smaller templates simply require smaller numbers of repetitive patterns: ▫:4; ⊞:7; ▫▫:17; ▫▫:11; ▫▫:12. To extend the class of systems considered in the main text, one can increase the size of the templates, or increase the number of possible colors for each cell. For 3×3 templates with two colors extensive randomized searches have failed to discover examples where non-repetitive patterns are forced to occur. Another extension of the constraints in the main text is to require that not just a single template, but every template in the set, must occur somewhere in the pattern. Searches of such systems have also failed to discover examples of forced non-repetitive patterns beyond the one shown in the text.

■ **Forcing nested patterns.** It is straightforward to find constraints that allow nested patterns; the challenge is to find ones that force such patterns to occur. Many nested patterns (such as the one made by rule 90, for example) contain large areas of uniform white, and it is typically difficult to prevent pure repetition of that area. One approach to finding constraints that can be satisfied only by nested patterns is nevertheless to start from specific nested patterns, look at what templates occur, and then see whether these templates are such that they do not allow any purely repetitive patterns. A convenient way to generate a large class of nested patterns is to use 2D substitution systems of the kind discussed on page 188. But searching all 4 billion or so possible such systems with 2×2 blocks and up to four colors one finds not a single case in which a nested pattern is forced to occur. It can nevertheless be shown that with a sufficiently large number of extra colors any nested pattern can be forced to occur. And it turns out that a result from the mid-1970s by Robert Ammann for a related problem of tiling (see below) allows one to construct a specific system with 16 colors in which constraints of the kind discussed here force a nested pattern to occur. One starts from the substitution system with rules

```
{1 → {{3}}, 2 → {{13, 1}, {4, 10}}, 3 → {{15, 1}, {4, 12}},
  4 → {{14, 1}, {2, 9}}, 5 → {{13, 1}, {4, 12}}, 6 → {{13, 1}, {8, 9}},
  7 → {{15, 1}, {4, 10}}, 8 → {{14, 1}, {6, 10}}, 9 → {{14}, {2}},
  10 → {{16}, {7}}, 11 → {{13}, {8}}, 12 → {{16}, {3}},
  13 → {{5, 11}}, 14 → {{2, 9}}, 15 → {{3, 11}}, 16 → {{6, 10}}}
```

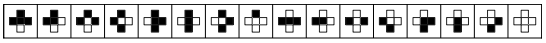
This yields the nested pattern below which contains only 51 of the 65,536 possible 2×2 blocks of cells with 16 colors. It then turns out that with the constraint that the only 2×2 arrangements of colors that can occur are ones that match these 51 blocks, one is forced to get the nested pattern below.



■ **Relation to 2D cellular automata.** The kind of constraints discussed are exactly those that must be satisfied by configurations that remain unchanged in the evolution of a 2D cellular automaton. The argument for this is similar to the one on pages 941 and 954 for 1D cellular automata. The point is that of the 32 5-cell neighborhoods involved in the 2D cellular automaton rule, only some subset will have the property that the center cell remains unchanged after applying the rule. And any configuration which does not change must involve only these subsets. Using the results of this section it then follows that in the evolution of all 2D

cellular automata of the type discussed on page 170 there exist purely repetitive configurations that remain unchanged.

▪ **Relation to 1D cellular automata.** A picture that shows the evolution of a 1D cellular automaton can be thought of as a 2D array of cells in which the color of each cell satisfies a constraint that relates it to the cells above according to the cellular automaton rule. This constraint can then be represented in terms of a set of allowed templates; the set for rule 30 is as follows:



To reproduce an ordinary picture of cellular automaton evolution, one would have to specify in advance a whole line of black and white cells. Below this line there would then be a unique pattern corresponding to the application of the cellular automaton rule. But above the line, except for reversible rules, there is no guarantee that any pattern satisfying the constraints can exist.

If one specifies no cells in advance, or at most a few cells, as in the systems discussed in the main text, then the issue is different, however. And now it is always possible to construct a repetitive pattern which satisfies the constraints simply by finding repetitive behavior in the evolution of the cellular automaton from a spatially repetitive initial condition.

▪ **Non-computable patterns.** It is known to be possible to set up constraints that will force patterns in which finding the color of a particular cell can require doing something like solving a halting problem—which cannot in general be done by any finite computation. (See also page 1139.)

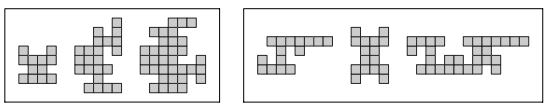
▪ **Tiling.** The constraints discussed here are similar to those encountered in covering the plane with tiles of various shapes. Of regular polygons, only squares, triangles and hexagons can be used to do this, and in these cases the tilings are always repetitive. For some time it was believed that any set of tiles that could cover the plane could be arranged to do so repetitively. But in 1964 Robert Berger demonstrated that this was not the case, and constructed a set of about 20,000 tiles that could cover the plane only in a nested fashion. Later Berger reduced the number of tiles needed to 104. Then Raphael Robinson in 1971 reduced the number tiles to six, and in 1974 Roger Penrose showed that just two tiles were necessary. Penrose's tiles can cover the plane only in a nested pattern that can be constructed from a substitution system that successively subdivides each tile, as shown on page 932. (Note that various dissections of these tiles can also be used. The edges of the particular shapes shown should strictly be distinguished in order to prevent trivial periodic arrangements.) The triangles in the construction have angles which are multiples of $\pi/5$, so that the whole tiling has an approximate 5-fold symmetry (see page 994). Repetitive tilings of the plane can only have 3-, 4- or 6-fold symmetry.

No single shape is known which has the property that it can tile the plane only non-repetitively, although one strongly suspects that one must exist. In 3D, John Conway has found a single biprism that can fill space only in a sequence of layers with an irrational rotation angle between each layer.

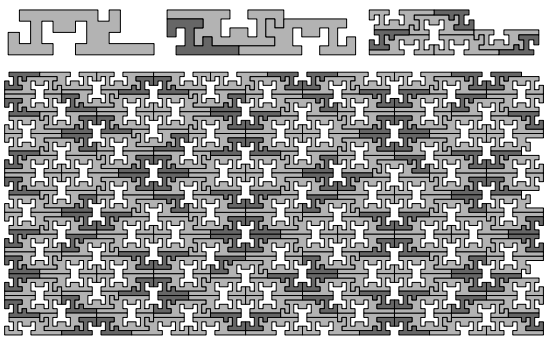
In addition, in no case has a simple set of tiles been found which force a pattern more complicated than a nested one. The results on page 221 in this book can be used to constructed a complicated set of tiles with this property, but I suspect that a much simpler set could be found.

(See also page 1139.)

▪ **Polyominoes.** An example of a tiling problem that is in some respects particularly close to the grid-based constraint systems discussed in the main text concerns covering the plane with polyominoes that are formed by gluing collections of squares together. Tiling by polyominoes has been investigated since at least the late 1950s, particularly by Solomon Golomb, but it is only very recently that sets of polyominoes which force non-periodic patterns have been found. The set (a) below was announced by Roger Penrose in 1994; the slightly smaller set (b) was found by Matthew Cook as part of the development of this book.



Both of these sets yield nested patterns. Steps in the construction of the pattern for set (b) are shown below. At stage n the number of polyominoes of each type is $Fibonacci[2\,n-\{2,\,0,\,1\}]/\{1,\,2,\,1\}$. Set (a) works in a roughly similar way, but with a considerably more complicated recursion.



■ **Ground states of spin systems.** The constraints discussed in the main text are similar to those that arise in the physics of 2D spin systems. An example of such a system is the so-called Ising model discussed on page 981. The idea in all such systems is to have an array of spins, each of which can be either up or down. The energy associated with each spin is then given by some function which depends on the configuration of neighboring spins. The ground state of the system corresponds to an arrangement of spins with the smallest total energy. In the ordinary Ising model, this ground state is simply all spins up or all spins down. But in generalizations of the Ising model with more complicated energy functions, the conditions to get a state of the lowest possible energy can correspond exactly to the constraints discussed in the main text. And from the results shown one sees that in some cases random-looking ground states should occur. Note that a rather different way to get a somewhat similar ground state is to consider a spin glass, in which the standard Ising model energy function is used, but multiplied by -1 or +1 at random for each spin.

■ **Correspondence systems.** For a discussion of a class of 1D systems based on constraints see page 757.

■ **Sequence equations.** Another way to set up 1D systems based on constraints is by having equations like *Flatten[{x, 1, x, 0, y}] === Flatten[{0, y, 0, y, x}]*, where each variable stands for a list. Fairly simple such equations can force fairly complicated results, although as discussed on page 1141 there are known to be limits to this complexity.

■ **Pattern-avoiding sequences.** As another form of constraint one can require, say, that no pair of identical blocks ever appear together in a sequence, so that the sequence does not match *{___, x__, x__, ___}*. With just two possible elements, no sequence above length 3 can satisfy this constraint. But with *k = 3* possible elements, there are infinite nested sequences that can, such as the one produced by the substitution system *{0 → {0, 1, 2}, 1 → {0, 2}, 2 → {1}}*, starting with *{0}*. One can find the sequences of length *n* that work by using

```
Nest[DeleteCases[Flatten[Map[Table[Append[#, i - 1],
    {i, k}] &, #], 1], {___, x__, x__, ___}] &, {{}}, n]
```

and the number of these grows roughly like $3^{n/4}$.

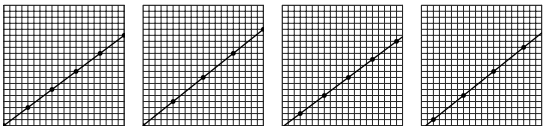
The constraint that no triple of identical blocks appear together turns out to be satisfied by the Thue-Morse nested sequence from page 83—as already noted by Axel Thue in 1906. (The number of sequences that work seems to grow roughly like $2^{n/2}$.)

For any given *k*, many combinations of blocks will inevitably occur in sufficiently long sequences (compare page 1068). (For example, with *k = 2*, *{___, x__, y__, x__, y__, ___}* always matches any sequence with length more than 18.) But some patterns of blocks can be avoided. And for example it is known that for $k \geq 2$ any pattern with length 6 or more (excluding the ___'s) and only two different variables (say *x__* and *y__*) can always be avoided. But it also known that among the infinite sequences which do this, there are always nested ones (sometimes one has to iterate one substitution rule, then at the end apply once a different substitution rule). With more variables, however, it seems possible that there will be patterns that can be avoided only by sequences with a more complicated structure. And a potential sign of this would be patterns for which the number of sequences that avoid them varies in a complicated way with length.

■ **Formal languages.** Formal languages of the kind discussed on page 938 can be used to define constraints on 1D sequences. The constraints shown on page 210 correspond to special cases of regular languages (see page 940). For both regular and context-free languages the so-called pumping lemmas imply that if any finite sequences satisfy the constraints, then so must an essentially repetitive infinite sequence.

■ **Diophantine equations.** Any algebraic equation—such as $x^3 + x + 1 == 0$—can readily be solved if one allows the variables to have any numerical value. But if one insists that the variables are whole numbers, then the problem is more analogous to the discrete constraints in the main text, and becomes much more difficult. And in fact, even though such so-called Diophantine equations have been studied since well before the time of Diophantus around perhaps 250 AD, only limited results about them are known.

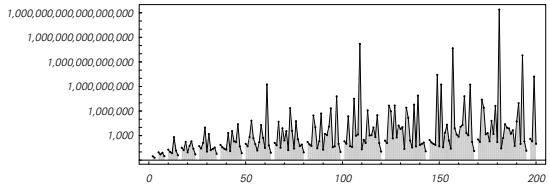
Linear Diophantine equations such as $a\,x == b\,y + c$ yield simple repetitive results, as in the pictures below, and can be handled essentially just by knowing *ExtendedGCD[a, b]*.



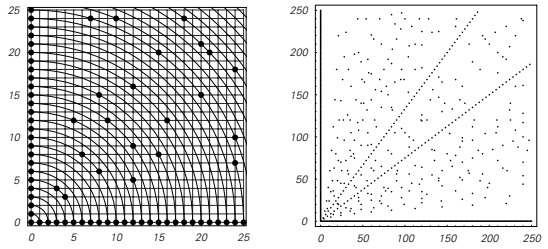
Even the simplest quadratic Diophantine equations can already show much more complex behavior. The equation $x^2 == a\,y^2$ has no solution except when *a* is a perfect square. But the Pell equation $x^2 == a\,y^2 + 1$ (already studied in antiquity) has infinitely many solutions whenever *a* is positive and not a perfect square. The smallest solution for *x* is given by

```
Numerator[FromContinuedFraction[
    ContinuedFraction[√a, (If[EvenQ[#], #, 2#] &)[
        Length[Last[ContinuedFraction[√a ]]]]]]]
```

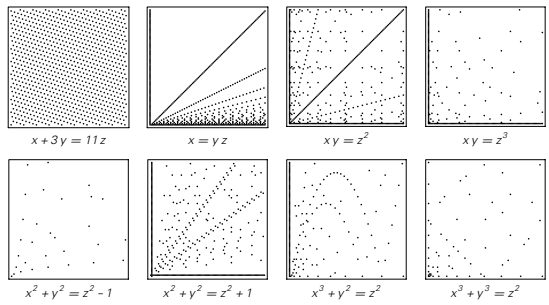
This is plotted below; complicated variation and some very large values are seen (with $a = 61$ for example $x == 1766319049$).



In three variables, the equation $x^2 + y^2 == z^2$ yields so-called Pythagorean triples $\{3, 4, 5\}$, $\{5, 12, 13\}$, etc. And even in this case the set of possible solutions for x and y in the pictures below looks fairly complicated—though after removing common factors, they are in fact just given by $\{x == r^2 - s^2, y == 2 r s, z == r^2 + s^2\}$. (See page 1078.)



The pictures below show the possible solutions for x and y in various Diophantine equations. As in other systems based on numbers, nested patterns are not common—though page 1160 shows how they can in principle be achieved with an equation whose solutions satisfy $Mod[Binomial[x, y], 2] == 1$. (The equation $(2x + 1)y == z$ also for example has solutions only when z is not of the form 2^j.)

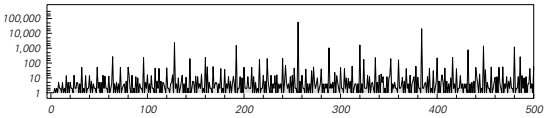


Many Diophantine equations have at most very sparse solutions. And indeed for example Fermat's Last Theorem states that $x^n + y^n == z^n$ can never be satisfied for $n > 2$. With four variables one has for example $3^3 + 4^3 + 5^3 == 6^3$, $1^3 + 6^3 + 8^3 == 9^3$—but with fourth powers the smallest result is $95800^4 + 217519^4 + 414560^4 == 422481^4$.

(See pages 791 and 1164.)

▪ **Matrices satisfying constraints.** One can consider for example magic squares, Latin squares (quasigroup multiplication tables), and matrices having the Hadamard property discussed on page 1073. One can also consider matrices whose powers contain certain patterns. (See also page 805.)

▪ **Finite groups and semigroups.** Any finite group or semigroup can be thought of as defined by having a multiplication table which satisfies the constraints given on page 887. The total number of semigroups increases faster than exponentially with size in a seemingly quite uniform way. But the number of groups varies in a complicated way with size, as in the picture below. (The peaks are known to grow roughly like $n\text{^}(2/27\ Log[2, n]^2)$—intermediate between polynomial and exponential.) As mentioned on page 938, through major mathematical effort, a complete classification of all finite so-called simple groups that in effect have no factors is known. Most such groups come in families that are easy to characterize; a handful of so-called sporadic ones are much more difficult to find. But this classification does not immediately provide a practical way to enumerate all possible groups. (See also pages 938 and 1032.)



▪ **Constraints on formulas.** Many standard problems of algebraic computation can be viewed as consisting in finding formulas that satisfy certain constraints. An example is exact solution of algebraic equations. For quadratic equations the standard formula gives solutions for arbitrary coefficients in terms of square roots. Similar formulas in terms of n^{th} roots have been known since the 1500s for equations with degrees n up to 4, although their *LeafCount* starting at $n = 1$ increases like 6, 25, 183, 718. For higher degrees it is known that such general formulas must involve other functions. For degrees 5 and 6 it was shown in the late 1800s that *EllipticTheta* or *Hypergeometric2F1* are sufficient, although for degrees 5 and 6 respectively the necessary formulas have a *LeafCount* in the billions. (Sharing common subexpressions yields a *LeafCount* in the thousands.) (See also page 1129.)

Starting from Randomness

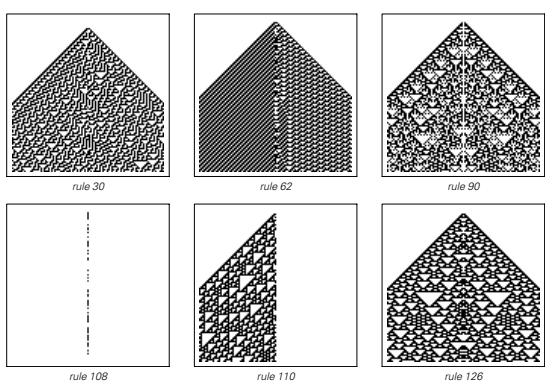
The Emergence of Order

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

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

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

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



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

▪ **History.** The fact that despite initial randomness processes like friction can make systems settle down into definite configurations has been the basis for all sorts of engineering throughout history. The rise of statistical mechanics in the late 1800s emphasized the idea of entropy increase and the fundamental tendency for systems to become progressively more disordered as they evolve to thermodynamic equilibrium. Theories were nevertheless developed for a few cases of spontaneous pattern formation—notably in convection, cirrus clouds and ocean waves. When the study of feedback and stability became popular in the 1940s, there were many results about how specific simple fixed or repetitive behaviors in time could emerge despite random input. In the 1950s it was suggested that reaction-diffusion processes might be responsible for spontaneous pattern formation in biology (see page 1012)—and starting in the 1970s such processes were discussed as prime examples of the phenomenon of self-organization. But in their usual form, they yield essentially only rather simple repetitive patterns. Ever since around 1900 it tended to be assumed that any fundamental theory of systems with many components must be based on statistical mechanics. But almost all work in the field of statistical mechanics concentrated on systems in or very near thermal equilibrium—in which in a sense there is almost complete disorder. In the 1970s there began to be more discussion of phenomena far from equilibrium, although typically it got no further than to consider how external forces could lead to reaction-diffusion-like phenomena. My own work on cellular

automata in 1981 emerged in part from thinking about self-gravitating systems (see page 880) where it seemed conceivable that there might be very basic rules quite different from those usually studied in statistical mechanics. And when I first generated pictures of the behavior of arbitrary cellular automaton rules, what struck me most was the order that emerged even from random initial conditions. But while it was immediately clear that most cellular automata do not have the kind of reversible underlying rules assumed in traditional statistical mechanics, it still seemed initially very surprising that their overall behavior could be so elaborate—and so far from the complete orderlessness one might expect on the basis of traditional ideas of entropy maximization.

Four Classes of Behavior

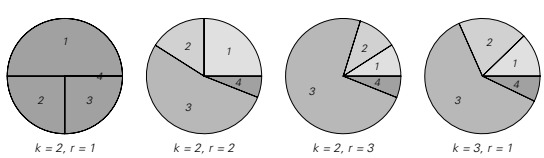
■ **Different runs.** The qualitative behavior seen with a given cellular automaton rule will normally look exactly the same for essentially all different large random initial conditions—just as it does for different parts of a single initial condition. And as discussed on page 597 any obvious differences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

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

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

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

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



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

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

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

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

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

■ **2D class 4 cellular automata.** No 5- or 9-neighbor totalistic rules nor 5-neighbor outer totalistic ones appear to yield

class 4 behavior with a white background. But among 9-neighbor outer totalistic rules there are examples with codes 224 (Game of Life), 226, 4320 (sometimes called HighLife), 5344, 6248, 6752, 6754 and 8416, etc. It turns out that the simplest moving structures are the same in codes 224, 226 and 4320.

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

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

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

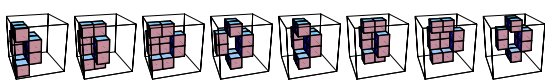
```
LifeStep[list_] :=
  With[{p = Flatten[Array[List, {3, 3}, -1], 1]},
    With[{u = Split[Sort[Flatten[Outer[Plus, list, p, 1], 1]]]},
      Union[Cases[u, {x_, _, _} → x],
        Intersection[Cases[u, {x_, _, _, _} → x], list]]]]
```

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

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

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

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



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

Sensitivity to Initial Conditions

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

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

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

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

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

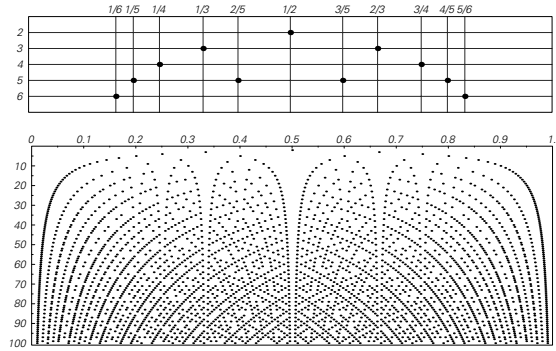
For any additive or partially additive class 3 cellular automaton (such as rule 90 or rule 30) any change in initial conditions will always lead to expanding differences. But in other rules it sometimes may not. And thus, for example, in rule 22, changing the color of a single cell has no effect after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

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

Systems of Limited Size and Class 2 Behavior

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

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



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

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

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

or equivalently

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

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

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

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

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

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

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

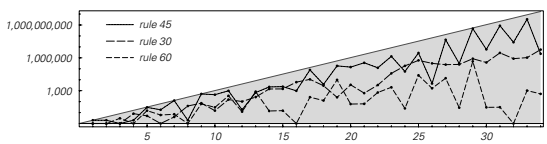
In the case of rule 90 a similar analysis can be given, with the $1 + x$ used at each step replaced by $1/x + x$. And now the repetition period for odd n divides

q[n] = 2^MultiplicativeOrder[2, n, {1, -1}] - 1

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

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

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

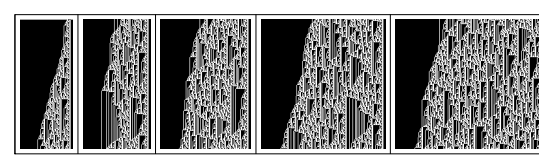


■ **Implementing boundary conditions.** In the bitwise representation discussed on page 865, 0's outside of a width n can be implemented by applying *BitAnd[a, 2^n - 1]* at each step. Cyclic boundary conditions can be implemented efficiently in assembler on computers that support cyclic shift instructions.

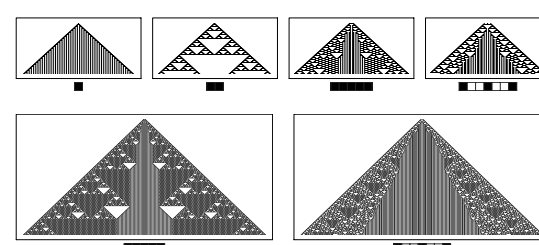
Randomness in Class 3 Systems

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

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



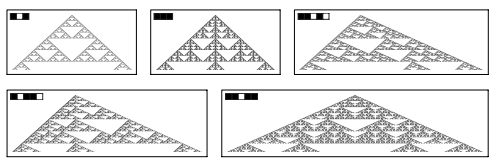
■ **Rule 94.** With appropriate initial conditions this class 2 rule can yield both nested and random behavior, as shown below.



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

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

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



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

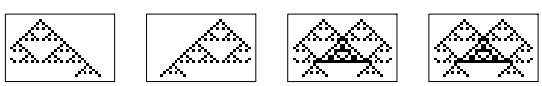
```
Mod[ListCorrelate[w, list, Ceiling[Length[w]/2]], k]
```

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

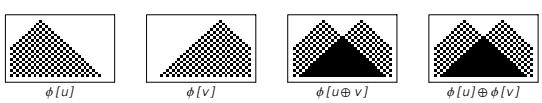
■ **Page 264 · Generalized additivity.** In general what it means for a system to be additive is that some addition operation $\oplus$ can be used to combine any set of evolution histories to yield another possible evolution history. If ϕ is the rule for the system, this then requires for any states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

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

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

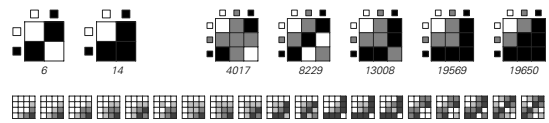


But it turns out that some elementary rules show additivity with respect to other addition operations. An example as shown below is rule 250 with $u \oplus v$ taken as *Max[u, v]* (*Or*).



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

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



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

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

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

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

If instead of an ordinary cellular automaton with a limited number of possible colors one considers a system in which every cell can have any integer value then additivity with respect to ordinary addition becomes just traditional linearity. And the only way to achieve this is to have a rule in which the new value of a cell is given by a linear form such as $a\,x + b\,y$. If the values of cells are allowed to be any real number then

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

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

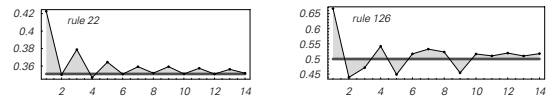
■ **Probabilistic estimates.** One way to get estimates for density and other properties of class 3 cellular automata is to make the assumption that the color of each cell at each step is completely random. And with this assumption, if the overall density of black cells at a particular step is p, then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

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

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

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

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



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

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

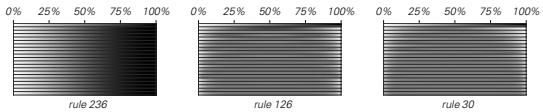
Probabilistic estimates can also be used for other quantities, such as growth rates of difference patterns (see page 949). In most cases, however, buildup of correlations tends to prevent systematic improvement of such approximations.

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

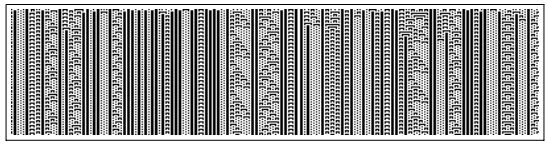
```
1/2 (1 - (1 - 2 p))^(2 ^ DigitCount[t, 2, 1])
```

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

Page 339 shows a cellular automaton with very different behavior.

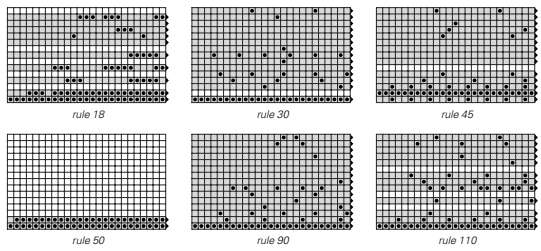


■ **Density oscillations in rule 73.** Although there are always some fluctuations, most rules yield densities that converge more or less uniformly to their final values. One exception is rule 73, which yields densities that continue to oscillate with a period of 3 steps forever. The origin of this phenomenon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



Special Initial Conditions

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



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

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

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

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

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

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

(See also page 700.)

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

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

considers configurations in 2D that consist only of infinitely long stripes, then the problem reduces again to the 1D case. (See also page 349.)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Other cases include (see page 870):

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

d[{1, 1, 1}, 3] = Log[3, 6] ≃ 1.63

d[{1, 1, 1}, 5] = Log[5, 19] ≃ 1.83

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

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

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

so that the first few steps yield

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

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

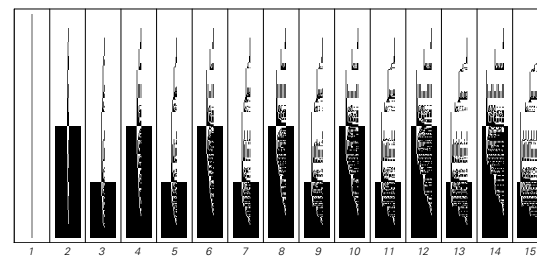
```
{f[0, 0, a], f[0, a, a, 0], f[a, 0, 0]}
{f[0, 0, 0, a], f[0, 0, a, 0, a, a, 0],
  f[0, a, a, 0, a, 0, 0], f[a, 0, 0, 0]}
```

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

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

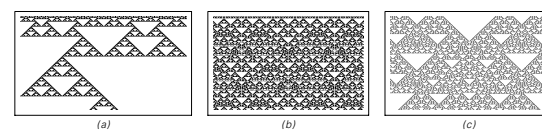
■ **Nesting in rule 45.** As illustrated on page 701, starting from a single black cell on a background of repeated ■□ blocks, rule 45 yields a slanted version of the nested rule 90 pattern.

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



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

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



The Notion of Attractors

■ **Page 275 · Discrete systems.** In traditional mathematics mechanical and other systems are assumed continuous, so that for example a pendulum may get exponentially close to

the attractor state where it has stopped, but it will never strictly reach this attractor. In discrete systems like cellular automata, however, there is no problem in explicitly reaching at least simple attractors.

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

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

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

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

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

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

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

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

this then yields for rule 126 the network

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

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

```
MinNet[net_, k_ : 2] := Module[{d = DSets[net, k], q, b},
 If[First[d] =!= {}, AllNet[k], q = ISets[b = Map[Table[
     Position[d, NetStep[net, #, a]][[1, 1]], {a, 0, k - 1}] &, d]];
  DeleteCases[MapIndexed[#2[[2]] - 1 → #1 &, Rest[
    Map[Position[q, #][[1, 1]] &, Transpose[Map[#[[Map[
      First, q]]] &, Transpose[b]]], {2}]] - 1, {2}], _ → 0, {2}]]]

DSets[net_, k_ : 2] :=
 FixedPoint[Union[Flatten[Map[Table[NetStep[net, #, a],
       {a, 0, k - 1}] &, #], 1]] &, {Range[Length[net]]}]

ISets[list_] := FixedPoint[Function[g, Flatten[Map[
   Map[Last, Split[Sort[Transpose[{Map[Position[g, #][[1,
     1]] &, list, {2}], Range[Length[list]]}][[#]]], First[#1] ==
    First[#2] &], {2}] &, g], 1]], {{1}, Range[2, Length[list]]}]
```

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

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

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

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

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

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

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

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

is always rational. (Compare the discussion of entropies below.)

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

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

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

and for rule 110 they are

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

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

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

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

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

where the adjacency matrix m is given by

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

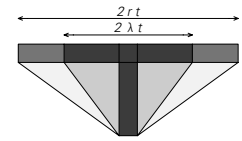
■ **Surjectivity and injectivity.** One can think of a cellular automaton rule as a mapping (endomorphism) from the space of possible states of the cellular automaton to itself. (See page 869.) Usually this mapping is contractive, so that not all the states which appear as input to the mapping can also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

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

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

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

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



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

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

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

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

■ **History.** The analysis of cellular automata given in this section is largely as I worked it out in the early 1980s. Parts of it, however, are related to earlier investigations, particularly in dynamical systems theory. Starting in the 1930s the idea of symbolic dynamics began to emerge, in which one partitions continuous values in a system into bins represented by discrete symbols, and then looks at the sequences of such symbols that can be produced by the evolution of the system. In connection with early work on

chaos theory, it was noted that there are some systems that act like "full shifts", in the sense that the set of sequences they generate includes all possibilities—and corresponds to what one would get by starting with any possible number, then successively shifting digits to the left, and at each step picking off the leading digit. It was noted that some systems could also yield various kinds of subshifts that are subsets of full shifts. But since—unlike in cellular automata—the symbol sequences being studied were obtained by rather arbitrary partitionings of continuous values, the question arose of what effect using different partitionings would have. One approach was to try to find invariants that would remain unchanged in different partitionings—and this is what led, for example, to the study of topological entropy in the 1960s. Another approach was to look at actual possible transformations between partitionings, and this led from the late 1950s to various studies of so-called shift-commuting block maps (or sliding-block codes)—which turn out to be exactly 1D cellular automata (see page 878). The locality of cellular automaton rules was thought of as making them the analog for symbol sequences of continuous functions for real numbers (compare page 869). Of particular interest were invertible (reversible) cellular automaton rules, since systems related by these were considered conjugate or topologically equivalent.

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

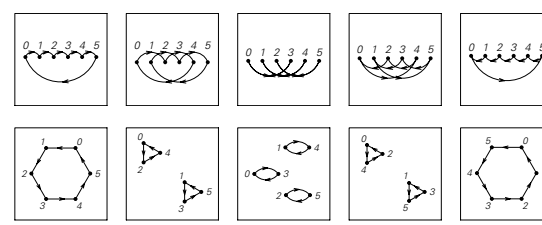
When I started working on cellular automata in the early 1980s I wanted to see how far one could get by following ideas of statistical mechanics and dynamical systems theory and trying to find global characterizations of the possible behavior of individual cellular automata. In the traditional symbolic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

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

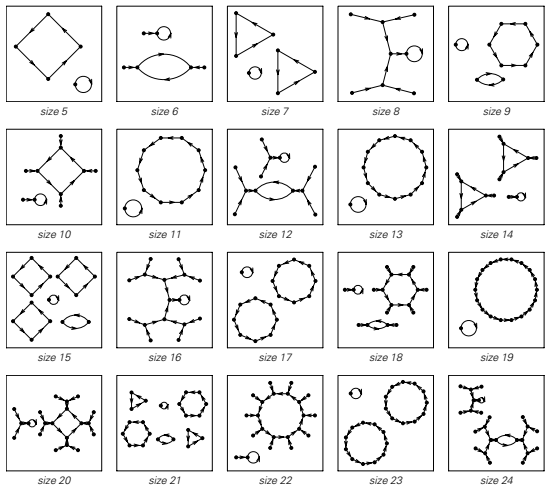
▪ **Iterated maps.** For maps of the form $x \to a\,x\,(1-x)$ discussed on page 920 the attractor for small a is a fixed point, then a period 2 limit cycle, then period 4, 8, 16, etc. There is an accumulation of limit cycles at $a \simeq 3.569946$ where the system has a special nested structure. (See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

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

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

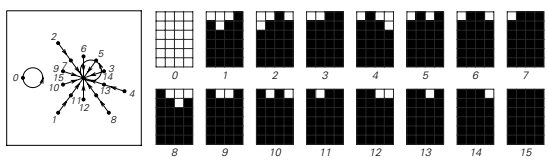


The pictures below give networks obtained from the system shown on page 257 for various values of n. For odd n, the networks consist purely of cycles. But for even n, there are also trees of states that lead to these cycles.

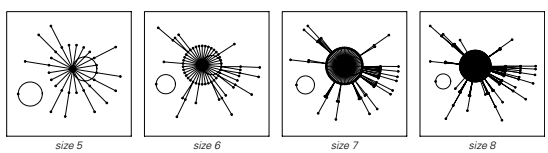


In general, any network that represents the evolution of a system with definite rules will have the same basic form. There are cycles which contain states that are visited repeatedly, and there can also be trees that represent transient states that can each only ever occur at most once in the evolution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

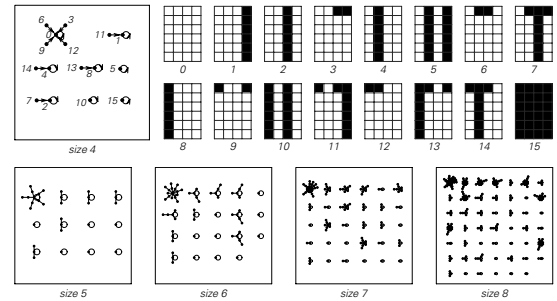
The picture below shows the network obtained from a class 1 cellular automaton (rule 254) with 4 cells and thus 16 possible states. All but one of these 16 states evolve after at most two steps to state 15, which corresponds to all cells being black.



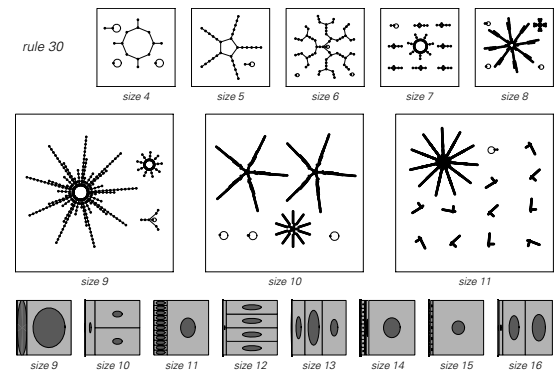
The pictures below show networks obtained when more cells are included in the cellular automaton above. The same convergence to a single fixed point is observed.

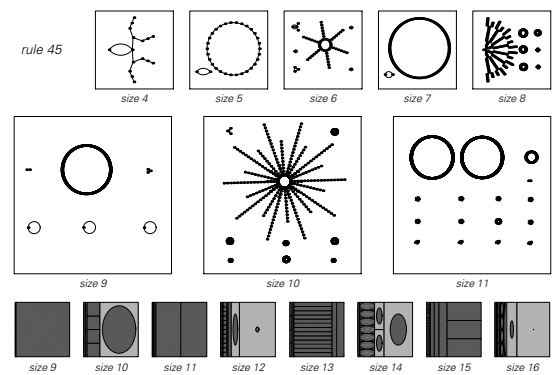


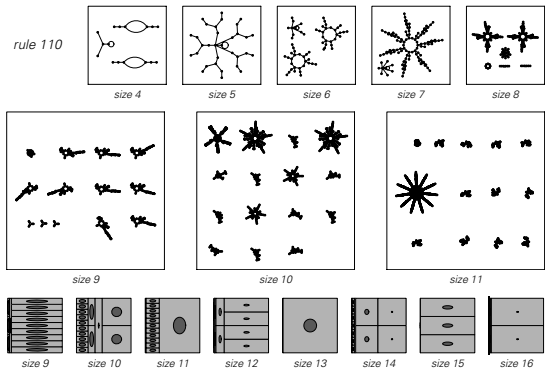
The pictures below give corresponding results for a class 2 cellular automaton (rule 132). The number of distinct cycles now increases with the size of the system. (As discussed below, identical pieces of the network are often related by symmetries of the underlying cellular automaton system.)



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







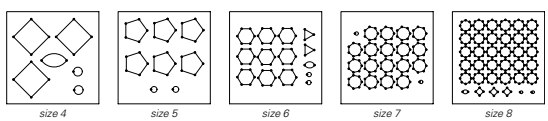
For large sizes there is a rough correspondence with the infinite size case, but many features are still different. (To recover correct infinite size results one must increase size while keeping the number of steps of evolution fixed; the networks shown above, however, effectively depend on arbitrarily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

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

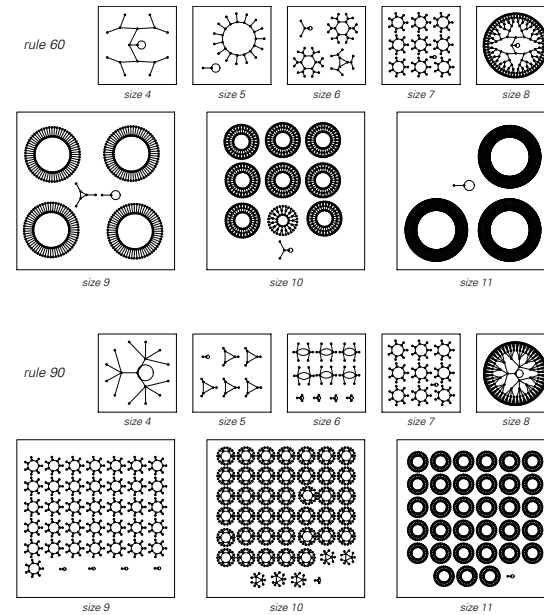
■ **Shift rules.** The pictures below show networks obtained with rule 170, which just shifts every configuration one position to the left at each step. With any such shift rule, all states lie on cycles, and the lengths of these cycles are the divisors of the size n. Every cycle corresponds in effect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

```
Apply[Plus, (EulerPhi[n/#] k^# &)[Divisors[n]]]/n
```

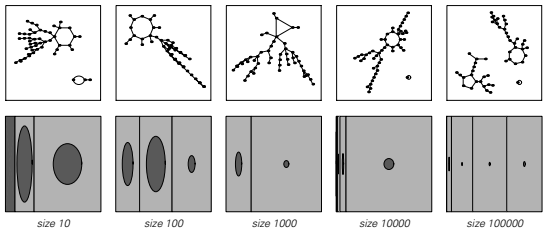
The number of cycles of length exactly m is $s[m, k]/m$, where $s[m, k]$ is defined on page 950. For prime k, each cycle (except all 0's) corresponds to a term in the product $Factor[x^{k^n-1} - 1, Modulus \to k]$. (See page 975.)



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

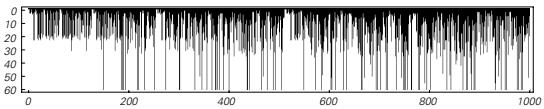


■ **Random networks.** The pictures below show networks in which each of a set of n nodes has as its successor a node that is chosen at random from the set. The total number of possible such networks is n^n. For large n, the average number of distinct cycles in all such networks is $Sqrt[\pi/2] Log[n]$,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

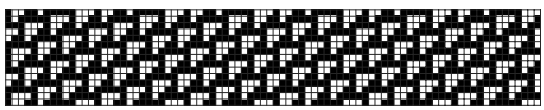


Structures in Class 4 Systems

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



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



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

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

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

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

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

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

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

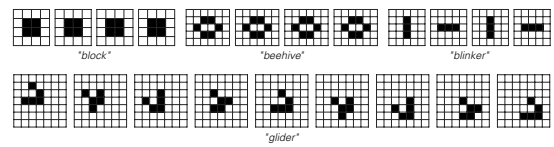
```
{1784, 801016, 410097400, 13304, 6406392, 3280778648}
```

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

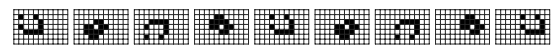
■ **Page 293 · Glider gun.** The initial conditions shown correspond to *{n, w} = {1339191737336, 41}*.

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

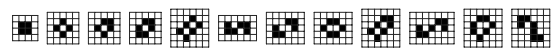
■ **The Game of Life.** The 2D cellular automaton described on page 949 supports a whole range of persistent structures, many of which have been given quaint names by its enthusiasts. With typical random initial conditions the most common structures to occur are:



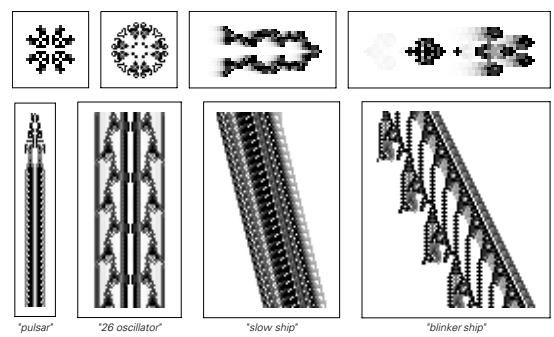
The next most common moving structure is the so-called "spaceship":



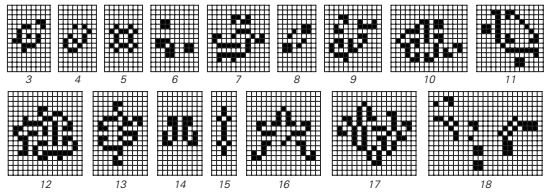
The complete set of structures with less than 8 black cells that remain unchanged at every step in the evolution are:



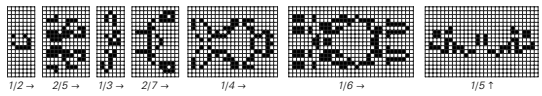
More complicated repetitive and moving structures are shown in the pictures below. If one looks at the history of a single row of cells, it typically looks much like the complete histories we have seen in 1D class 4 cellular automata.



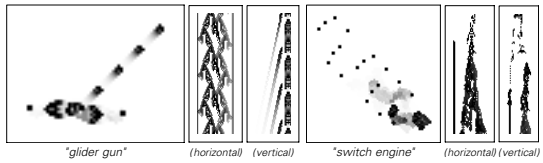
Structures with all repetition periods up to 18 have been found in Life; examples are shown in the pictures below.



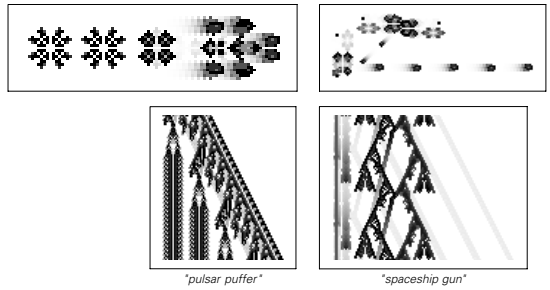
Persistent structures with various speeds in the horizontal and vertical direction have also been found, as shown below.



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

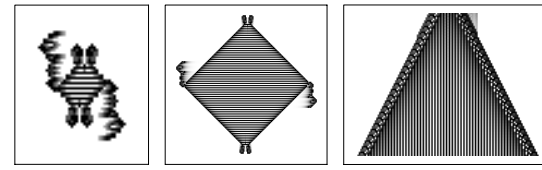


Many more elaborate structures similar to the glider gun were found in the 1970s and 1980s; two are illustrated below.

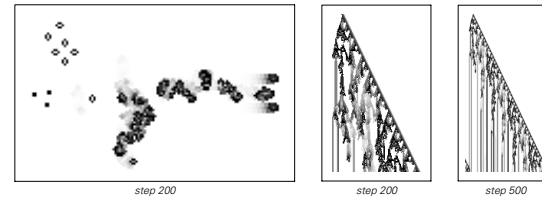


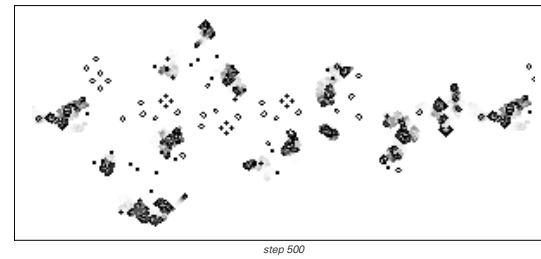
A simpler kind of unbounded growth occurs if one starts from an infinite line of black cells. In that case, the evolution is effectively 1D, and turns out to follow elementary rule 22, thus producing the infinitely growing nested pattern shown on page 263.

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



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





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

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

Mechanisms in Programs and Nature

Universality of Behavior

■ **History.** That very different natural and artificial systems can show similar forms has been noted for many centuries. Informal studies have been done by a whole sequence of architects interested both in codifying possible forms and in finding ways to make structures fit in with nature and with our perception of it. Beginning in the Renaissance the point has also been noted by representational and decorative artists, most often in the context of developing a theory of the types of forms to be studied by students of art. The growth of comparative anatomy in the 1800s led to attempts at more scientific treatments, with analogies between biological and physical systems being emphasized particularly by D'Arcy Thompson in 1917. Yet despite all this, the phenomenon of similarity between forms remained largely a curiosity, discussed mainly in illustrated books with no clear basis in either art or science. In a few cases (such as work by Peter Stevens in 1974) general themes were however suggested. These included for example symmetry, the golden ratio, spirals, vortices, minimal surfaces, branching patterns, and—since the 1980s—fractals. The suggestion is also sometimes made that we perceive a kind of harmony in nature because we see only a limited number of types of forms in it. And particularly in classical architecture the idea is almost universally used that structures will seem more comfortable to us if they repeat in ornament or otherwise forms with which we have become familiar from nature. Whenever a scientific model has the same character for different systems this means that the systems will tend to show similar forms. And as models like cellular automata capable of dealing with complexity have become more widespread it has been increasingly popular to show that they can capture similar forms seen in very different systems.

Three Mechanisms for Randomness

■ **Page 299 · Definition.** How randomness can be defined is discussed at length on page 552.

■ **History.** In antiquity, it was often assumed that all events must be governed by deterministic fate—with any apparent randomness being the result of arbitrariness on the part of the gods. Around 330 BC Aristotle mentioned that instead randomness might just be associated with coincidences outside whatever system one is looking at, while around 300 BC Epicurus suggested that there might be randomness continually injected into the motion of all atoms. The rise of emphasis on human free will (see page 1135) eroded belief in determinism, but did not especially address issues of randomness. By the 1700s the success of Newtonian physics seemed again to establish a form of determinism, and led to the assumption that whatever randomness was actually seen must reflect lack of knowledge on the part of the observer—or particularly in astronomy some form of error of measurement. The presence of apparent randomness in digit sequences of square roots, logarithms, numbers like π, and other mathematical constructs was presumably noticed by the 1600s (see page 911), and by the late 1800s it was being taken for granted. But the significance of this for randomness in nature was never recognized. In the late 1800s and early 1900s attempts to justify both statistical mechanics and probability theory led to ideas that perfect microscopic randomness might somehow be a fundamental feature of the physical world. And particularly with the rise of quantum mechanics it came to be thought that meaningful calculations could be done only on probabilities, not on individual random sequences. Indeed, in almost every area where quantitative methods were used, if randomness was observed, then either a different system was studied, or efforts were made to remove the randomness by averaging or some other statistical method. One case where there was occasional discussion of origins of randomness from at least

the early 1900s was fluid turbulence (see page 997). Early theories tended to concentrate on superpositions of repetitive motions, but by the 1970s ideas of chaos theory began to dominate. And in fact the widespread assumption emerged that between randomness in the environment, quantum randomness and chaos theory almost any observed randomness in nature could be accounted for. Traditional mathematical models of natural systems are often expressed in terms of probabilities, but do not normally involve anything one can explicitly consider as randomness. Models used in computer simulations, however, do very often use explicit randomness. For not knowing about the phenomenon of intrinsic randomness generation, it has normally been assumed that with the kinds of discrete elements and fairly simple rules common in such models, realistically complicated behavior can only ever be obtained if explicit randomness is continually introduced.

■ **Applications of randomness.** See page 1192.

■ **Sources of randomness.** Two simple mechanical methods for generating randomness seem to have been used in almost every civilization throughout recorded history. One is to toss an object and see which way up or where it lands; the other is to select an object from a collection mixed by shaking. The first method has been common in games of chance, with polyhedral dice already existing in 2750 BC. The second—often called drawing lots—has normally been used when there is more at stake. It is mentioned several times in the Bible, and even today remains the most common method for large lotteries. (See page 969.) Variants include methods such as drawing straws. In antiquity fortune-telling from randomness often involved looking say at growth patterns of goat entrails or sheep shoulder blades; today configurations of tea leaves are sometimes considered. In early modern times the matching of fracture patterns in broken tally sticks was used to identify counterparties in financial contracts. Horse races and other events used as a basis for gambling can be viewed as randomness sources. Children's games like musical chairs in effect generate randomness by picking arbitrary stopping times. Games of chance based on wheels seem to have existed in Roman times; roulette developed in the 1700s. Card shuffling (see page 974) has been used as a source of randomness since at least the 1300s. Pegboards (as on page 312) were used to demonstrate effects of randomness in the late 1800s. An explicit table of 40,000 random digits was created in 1927 by Leonard Tippett from details of census data. And in 1938 further tables were generated by Ronald Fisher from digits of logarithms. Several tables based on physical processes were produced, with the RAND Corporation in 1955 publishing a table of a million random digits obtained from an electronic roulette wheel. Beginning in the 1950s, however, it became increasingly common to use pseudorandom generators whenever long sequences were needed—with linear feedback shift registers being most popular in standalone electronic devices, and linear congruential generators in programs (see page 974). There nevertheless continued to be occasional work done on mechanical sources of randomness for toys and games, and on physical electronic sources for cryptography systems (see page 969).

Randomness from the Environment

■ **Page 301 · Stochastic models.** The mechanism for randomness discussed in this section is the basis for so-called stochastic models now widely used in traditional science. Typically the idea of these models is to approximate those elements of a system about which one does not know much by random variables. (See also page 588.) In the early work along these lines done by James Clerk Maxwell and others in the 1880s, analytical formulas were usually worked out for the probabilities of different outcomes. But when electronic computers became available in the 1940s, the so-called Monte Carlo method became increasingly popular, in which instead explicit simulations are performed with different choices of random variables, and then statistical averages are found. Early uses of the Monte Carlo method were mostly in physics, particularly for studies of neutron diffusion and particle shower generation in high-energy collisions. But by the 1980s the Monte Carlo method had also become common in other fields, and was routinely used in studying for example message flows in communication networks and pricing processes in financial markets. (See also page 1192.)

■ **Page 301 · Ocean surfaces.** See page 1001.

■ **Page 302 · Random walks.** See page 328.

■ **Page 302 · Electronic noise.** Three types of noise are commonly observed in typical devices:

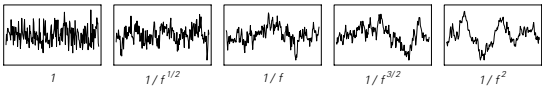
Shot noise. Electric currents are not continuous but are ultimately made up from large numbers of moving charge carriers, typically electrons. Shot noise arises from statistical fluctuations in the flow of charge carriers: if a single bit of data is represented by 10,000 electrons, the magnitude of the fluctuations will typically be about 1%. When looked at as a waveform over time, shot noise has a flat frequency spectrum.

Thermal (Johnson) noise. Even though an electric current may have a definite overall direction, the individual charge carriers within it will exhibit random motions. In a material

at nonzero temperature, the energy of these motions and thus the intensity of the thermal noise they produce is essentially proportional to temperature. (At very low temperatures, quantum mechanical fluctuations still yield random motion in most materials.) Like shot noise, thermal noise has a flat frequency spectrum.

Flicker (1/f) noise. Almost all electronic devices also exhibit a third kind of noise, whose main characteristic is that its spectrum is not flat, but instead goes roughly like $1/f$ over a wide range of frequencies. Such a spectrum implies the presence of large low-frequency fluctuations, and indeed fluctuations are often seen over timescales of many minutes or even hours. Unlike the types of noise described above, this kind of noise can be affected by details of the construction of individual devices. Although seen since the 1920s its origins remain somewhat mysterious (see below).

▪ **Power spectra.** Many random processes in nature show power spectra $Abs[Fourier[data]]^2$ with fairly simple forms. Most common are white noise uniform in frequency and $1/f^2$ noise associated with random walks. Other pure power laws $1/f^\alpha$ are also sometimes seen; the pictures below show some examples. (Note that the correlations in such data in some sense go like $t^{\alpha-1}$.) Particularly over the past few decades all sorts of examples of "$1/f$ noise" have been identified with $\alpha \simeq 1$, including flicker noise in resistors, semiconductor devices and vacuum tubes, as well as thunderstorms, earthquake and sunspot activity, heartbeat intervals, road traffic density and some DNA sequences. A pure $1/f^\alpha$ spectrum presumably reflects some form of underlying nesting or self-similarity, although exactly what has usually been difficult to determine. Mechanisms that generally seem able to give $\alpha \simeq 1$ include random walks with exponential waiting times, power-law distributions of step sizes (Lévy flights), or white noise variations of parameters, as well as random processes with exponentially distributed relaxation times (as from Boltzmann factors for uniformly distributed barrier heights), fractional integration of white noise, intermittency at transitions to chaos, and random substitution systems. (There was confusion in the late 1980s when theoretical studies of self-organized criticality failed correctly to take squares in computing power spectra.) Note that the Weierstrass function of page 918 yields a $1/f$ spectrum, and presumably suitable averages of spectra from any substitution system should also have $1/f^\alpha$ forms (compare page 586).



▪ **Page 303 · Spark chambers.** The sensitivity of sparks to microscopic details of the environment is highlighted by the several devices which essentially use them to detect the passage of individual elementary particles such as protons. Such particles leave a tiny trail of ionized gas, which becomes the path of the spark. This principle was used in Geiger counters, and later in spark chambers and wire chambers.

▪ **Physical randomness generators.** It is almost universally assumed that at some level physical processes must be the best potential sources of true randomness. But in practice their record has actually been very poor. It does not help that unlike algorithms physical devices can be affected by their environment, and can also not normally be copied identically. But in almost every case I know where detailed analysis has been done substantial deviations from perfect randomness have been found. This has however typically been attributed to engineering mistakes—or to sampling data too quickly—and not to anything more fundamental that is for example worth describing in publications.

▪ **Mechanical randomness.** It takes only small imperfections in dice or roulette wheels to get substantially non-random results (see page 971). Gaming regulations typically require dice to be perfect cubes to within one part in a few thousand; casinos normally retire dice after a few hundred rolls.

In processes like stirring and shaking it can take a long time for correlations to disappear—as in the phenomenon of long-time tails mentioned on page 999. One notable consequence were traces of insertion order among the 366 capsules used in the 1970 draft lottery in the U.S. But despite such problems mixing of objects remains by far the most common way to generate randomness when there is a desire for the public to see randomization occur. And so for example all the state lotteries in the U.S. are currently based on mixing between 10 and 54 balls. (Numbers games were instead sometimes based on digits of financial data in newspapers.)

There have been a steady stream of inventions for mechanical randomness generation. Some are essentially versions of dice. Others involve complicated cams or linkages, particularly for mechanical toys. And still others involve making objects like balls bounce around as randomly as possible in air or other fluids.

▪ **Electronic randomness.** Since the 1940s a steady stream of electronic devices for producing randomness have been invented, with no single one ever becoming widely used. An early example was the ERNIE machine from 1957 for British national lottery (premium bond) drawings, which worked by sampling shot noise from neon discharge

tubes—and perhaps because it extracted only a few digits per second no deviations from randomness in its output were found. (U.S. missiles apparently used a similar method to produce randomly spaced radar pulses for determining altitude.) Since the 1970s electronic randomness generators have typically been based on features of semiconductor devices—sometimes thermal noise, but more often breakdown, often in back-biased zener diodes. All sorts of schemes have been invented for getting unbiased output from such systems, and acceptable randomness can often be obtained at kilohertz rates, but obvious correlations almost always appear at higher rates. Macroscopic thermal diffusion undoubtedly underestimates the time for good microscopic randomization. For in addition to *1/f* noise effects, solitons and other collective lattice effects presumably lead to power-law decay of correlations. It still seems likely however that some general inequalities should exist between the rate and quality of randomness that can be extracted from a system with particular thermodynamic properties.

■ **Quantum randomness.** It is usually assumed that even if all else fails a quantum process such as radioactive decay will yield perfect randomness. But in practice the most accurate measurements show phenomena such as *1/f* noise, presumably as a result of features of the detector and perhaps of electromagnetic fields associated with decay products. Acceptable randomness has however been obtained at rates of tens of bits per second. Recent attempts have also been made to produce quantum randomness at megahertz rates by detecting paths of single photons. (See also page 1064.)

■ **Randomness in computer systems.** Most randomness needed in practical computer systems is generated purely by programs, as discussed on page 317. But to avoid having a particular program give exactly the same random sequence every time it is run, one usually starts from a seed chosen on the basis of some random feature of the environment. Until the early 1990s this seed was most often taken from the exact time of day indicated by the computer's clock at the moment when it was requested. But particularly in environments where multiple programs can start almost simultaneously other approaches became necessary. Versions of the Unix operating system, for example, began to support a virtual device (typically called /dev/random) to maintain a kind of pool of randomness based on details of the computer system. Most often this uses precise timings between interrupts generated by keys being pressed, a mouse being moved, or data being delivered from a disk, network, or other device. And to prevent the same state being reached every time a computer is rebooted, some information is permanently maintained in a file. At the end of the 1990s standard microprocessors also began to include instructions to sample thermal noise from an on-chip resistor. (Any password or encryption key made up by a human can be thought of as a source of randomness; some systems look at details of biometric data, or scribbles drawn with a mouse.)

■ **Randomness in biology.** Thermal fluctuations in chemical reactions lead to many kinds of microscopic randomness in biological systems, sometimes amplified when organisms grow. For example, small-scale randomness in embryos can affect large-scale pigmentation patterns in adult organisms, as discussed on page 1013. Random changes in single DNA molecules can have global effects on the development of an organism. Standard mitotic cell division normally produces identical copies of DNA—with random errors potentially leading for example to cancers. But in sexual reproduction genetic material is rearranged in ways normally assumed by classical genetics to be perfectly random. One reason is that which sperm fertilizes a given egg is determined by random details of sperm and fluid motion. Another reason is that egg and sperm cells get half the genetic material of an organism, somewhat at random. In most cells, say in humans, there are two versions of all 23 chromosomes—one from the father and one from the mother. But when meiosis forms egg and sperm cells they get only one version of each. There is also exchange of DNA between paternal and maternal chromosomes, typically with a few crossovers per chromosome, at positions that seem more or less randomly distributed among many possibilities (the details affect regions of repeating DNA used for example in DNA fingerprinting).

In the immune system blocks of DNA—and joins between them—are selected at random by microscopic chemical processes when antibodies are formed.

Most animal behavior is ultimately controlled by electrical activity in nerve cells—and this can be affected by details of sensory input, as well as by microscopic chemical processes in individual cells and synapses (see page 1011).

Flagellated microorganisms can show random changes in direction as a result of tumbling when their flagella counter-rotate and the filaments in them flail around.

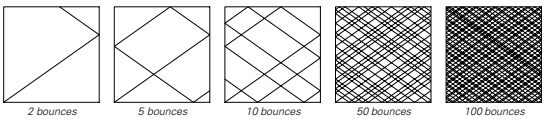
(See also page 1011.)

Chaos Theory and Randomness from Initial Conditions

■ **Page 305 · Spinning and tossing.** Starting with speed v, the speed of the ball at time t is simply $v - at$, where a is the deceleration produced by friction. The ball thus stops at time

v/a. The distance gone by the ball at a given time is $x = v\,t - a\,t^2/2$, and its orientation is $Mod[x, 2\,\pi\,r]$. For dice and coins there are some additional detailed effects associated with the shapes of these objects and the way they bounce. (Polyhedral dice have become more common since Dungeons & Dragons became popular in the late 1970s.) Note that in practice a coin tossed in the air will typically turn over between ten and twenty times while a die rolled on a table will turn over a few tens of times. A coin spun on a table can rotate several hundred times before falling over and coming to rest.

▪ **Billiards.** A somewhat related system is formed by a billiard ball bouncing around on a table. The issue of which sequence of horizontal and vertical sides the ball hits depends on the exact slope with which the ball is started (in the picture below it is $1/\sqrt{2}$). In general, it is given by the successive terms in the continued fraction form (see page 914) of this slope, and is related to substitution systems (see page 903). (See also page 1022.)



▪ **Fluttering.** If one releases a stationary piece of paper in air, then unlike a coin, it does not typically maintain the same orientation as it falls. Small pieces of paper spin in a repetitive way; but larger pieces of paper tend to flutter in a seemingly random way (as discussed, among others, by James Clerk Maxwell in 1853). A similar phenomenon can be seen if one drops a coin in water. I suspect that in these cases the randomness that occurs has an intrinsic origin, rather than being the result of sensitive dependence on initial conditions.

▪ **History of chaos theory.** The idea that small causes can sometimes have large effects has been noted by historians and others since antiquity, and captured for example in "for want of a nail … a kingdom was lost". In 1860 James Clerk Maxwell discussed how collisions between hard sphere molecules could lead to progressive amplification of small changes and yield microscopic randomness in gases. In the 1870s Maxwell also suggested that mechanical instability and amplification of infinitely small changes at occasional critical points might explain apparent free will (see page 1135). (It was already fairly well understood that for example small changes could determine which way a beam would buckle.) In 1890 Henri Poincaré found sensitive dependence on initial conditions in a particular case of the three-body problem (see below), and later proposed that such phenomena could be common, say in meteorology. In 1898 Jacques Hadamard noted general divergence of trajectories in spaces of negative curvature, and Pierre Duhem discussed the possible general significance of this in 1908. In the 1800s there had been work on nonlinear oscillators—particularly in connection with models of musical instruments—and in 1927 Balthazar van der Pol noted occasional "noisy" behavior in a vacuum tube oscillator circuit presumably governed by a simple nonlinear differential equation. By the 1930s the field of dynamical systems theory had begun to provide characterizations of possible forms of behavior in differential equations. And in the early 1940s Mary Cartwright and John Littlewood noted that van der Pol's equation could exhibit solutions somehow sensitive to all digits in its initial conditions. The iterated map $x \to 4\,x\,(1 - x)$ was also known to have a similar property (see page 918). But most investigations centered on simple and usually repetitive behavior—with any strange behavior implicitly assumed to be infinitely unlikely. In 1962, however, Edward Lorenz did a computer simulation of a set of simplified differential equations for fluid convection (see page 998) in which he saw complicated behavior that seemed to depend sensitively on initial conditions—in a way that he suggested was like the map $x \to FractionalPart[2\,x]$. In the mid-1960s, notably through the work of Steve Smale, proofs were given that there could be differential equations in which such sensitivity is generic. In the late 1960s there began to be all sorts of simulations of differential equations with complicated behavior, first mainly on analog computers, and later on digital computers. Then in the mid-1970s, particularly following discussion by Robert May, studies of iterated maps with sensitive dependence on initial conditions became common. Work by Robert Shaw in the late 1970s clarified connections between information content of initial conditions and apparent randomness of behavior. The term "chaos" had been used since antiquity to describe various forms of randomness, but in the late 1970s it became specifically tied to the phenomenon of sensitive dependence on initial conditions. By the early 1980s at least indirect signs of chaos in this sense (see note below) had been seen in all sorts of mechanical, electrical, fluid and other systems, and there emerged a widespread conviction that such chaos must be the source of all important randomness in nature. So in 1985 when I raised the possibility that intrinsic randomness might instead be a key phenomenon this was greeted with much hostility by some younger proponents of chaos theory. Insofar as what they had to say was of a scientific nature, their main point was that somehow what I

had seen in cellular automata must be specific to discrete systems, and would not occur in the continuous systems assumed to be relevant in nature. But from many results in this book it is now clear that this is not correct. (Note that James Gleick's 1987 popular book *Chaos* covers somewhat more than is usually considered chaos theory—including some of my results on cellular automata from the early 1980s.)

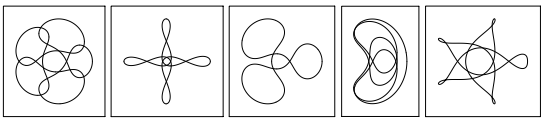
■ **Information content of initial conditions.** See page 920.

■ **Recognizing chaos.** Any system that depends sensitively on digits in its initial conditions must necessarily be able to show behavior that is not purely repetitive (compare page 955). And when it is said that chaos has been found in a particular system in nature what this most often actually means is just that behavior with no specific repetition frequency has been seen (compare page 586). To give evidence that this is not merely a reflection of continual injection of randomness from the environment what is normally done is to show that at least some aspect of the behavior of the system can be fit by a definite simple iterated map or differential equation. But inevitably the fit will only be approximate, so there will always be room for effects from randomness in the environment. And in general this kind of approach can never establish that sensitive dependence on initial conditions is actually the dominant source of randomness in a given system—say as opposed to intrinsic randomness generation. (Attempts are sometimes made to detect sensitive dependence directly by watching whether a system can do different things after it appears to return to almost exactly the same state. But the problem is that it is hard to be sure that the system really is in the same state—and that there are not all sorts of large differences that do not happen to have been observed.)

■ **Instability.** Sensitive dependence on initial conditions is associated with a kind of uniform instability in systems. But vastly more common in practice is instability only at specific critical points—say bifurcation points—combined with either intrinsic randomness generation or randomness from the environment. (Note that despite its widespread use in discussions of chaos theory, this is also what usually seems to happen with the weather; see page 1177.)

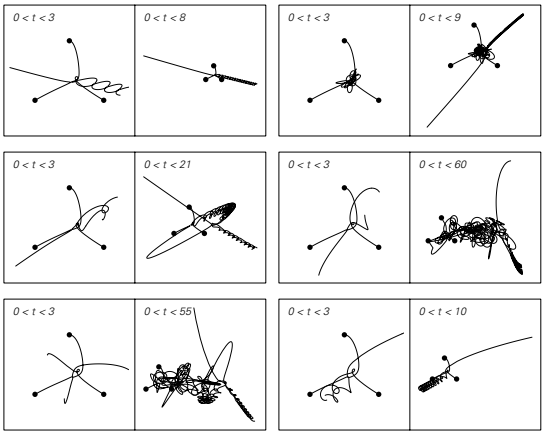
■ **Page 313 · Three-body problem.** The two-body problem was analyzed by Johannes Kepler in 1609 and solved by Isaac Newton in 1687. The three-body problem was a central topic in mathematical physics from the mid-1700s until the early 1900s. Various exact results were obtained—notably the existence of stable equilateral triangle configurations corresponding to so-called Lagrange points. Many approximate practical calculations, particularly on the Earth-Moon-Sun system, were done using series expansions involving thousands of algebraic terms. (It is now possible to get most results just by direct numerical computation using for example *NDSolve*.) From its basic setup the three-body system conserves standard mechanical quantities like energy and angular momentum. But it was thought it might also conserve other quantities (or so-called integrals of the motion). In 1887, however, Heinrich Bruns showed that there could be no such quantities expressible as algebraic functions of the positions and velocities of the bodies (in standard Cartesian coordinates). In the mid-1890s Henri Poincaré then showed that there could also be no such quantities analytic in positions, velocities and mass ratios. And from these results the conclusion was drawn that the three-body problem could not be solved in terms of algebraic formulas and integrals. In 1912 Karl Sundman did however find an infinite series that could in principle be summed to give the solution—but which converges exceptionally slowly. And even now it remains conceivable that the three-body problem could be solved in terms of more sophisticated standard mathematical functions. But I strongly suspect that in fact nothing like this will ever be possible and that instead the three-body problem will turn out to show the phenomenon of computational irreducibility discussed in Chapter 12 (and that for example three-body systems are universal and in effect able to perform any computation). (See also page 1132.)

In Henri Poincaré's study of the collection of possible trajectories for three-body systems he identified sensitive dependence on initial conditions (see above), noted the general complexity of what could happen (particularly in connection with so-called homoclinic tangles), and developed topology to provide a simpler overall description. With appropriate initial conditions one can get various forms of simple behavior. The pictures below show some of the possible repetitive orbits of an idealized planet moving in the plane of a pair of stars that are in a perfect elliptical orbit.



The pictures below show results for a fairly typical sequence of initial conditions where all three bodies interact. (The two bodies at the bottom are initially at rest; the body at the top is given progressively larger rightward velocities.) What generically happens is that one of the bodies escapes from the other two (like t or sometimes $t^{2/3}$). Often this happens quickly, but sometimes all three bodies show complex and

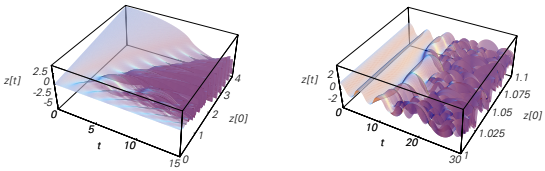
apparently random behavior for quite a while. (The delay before escaping is reminiscent of resonant scattering.)



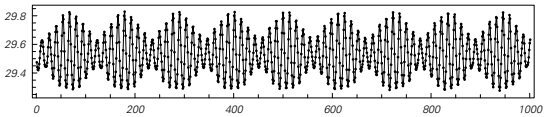
▪ **Page 314 · Simple case.** The position of the idealized planet in the case shown satisfies the differential equation

$$\partial_{tt} z[t] == -z[t]/(z[t]^2 + (1/2\,(1 + e\,\mathrm{Sin}[2\,\pi\,t]))^2)^{3/2}$$

where e is the eccentricity of the elliptical orbit of the stars ($e = 0.1$ in the picture). (Note that the physical situation is unstable: if the planet is perturbed so that there is a difference between its distance to each star, this will tend to increase.) Except when $e = 0$, the equation has no solution in terms of standard mathematical functions. It can be solved numerically in *Mathematica* using *NDSolve*, although a working precision of 40 decimal digits was used to obtain the results shown. Following work by Kirill Sitnikov in 1960 and by Vladimir Alekseev in 1968, it was established that with suitably chosen initial conditions, the equation yields any sequence *Floor[t[i + 1] - t[i]]* of successive zero-crossing times *t[i]*. The pictures below show the dependence of *z[t]* on *t* and *z[0]*. As *t* increases, *z[t]* typically begins to vary more rapidly with *z[0]*—reflecting sensitive dependence on initial conditions.



▪ **Page 314 · Randomness in the solar system.** Most motion observed in the solar system on human timescales is highly regular—though sometimes intricate, as in the sequence of numbers of days between successive new moons shown below. In the mid-1980s, however, work by Jack Wisdom and others established that randomness associated with sensitive dependence on initial conditions could occur in certain current situations in the solar system, notably in the orbits of asteroids. Various calculations suggest that there should also be sensitive dependence on initial conditions in the orbits of planets in the solar system—with effects doubling every few million years. But there are so far no observational signs of randomness resulting from this, and indeed the planets—at least now—mostly just seem to have orbits that are within a few percent of circles. If a planet moved in too random a way then it would tend to collide or escape from the solar system. And indeed it seems quite likely that in the past there may have been significantly more planets in our solar system—with only those that maintained regular orbits now being left. (See also page 1021.)



The Intrinsic Generation of Randomness

▪ **Autoplectic processes.** In the 1985 paper where I introduced intrinsic randomness generation I called processes that show this autoplectic, while I called processes that transcribe randomness from outside homoplectic.

▪ **Page 316 · Algorithmic randomness.** The idea of there being no simple procedure that can generate a particular sequence can be stated more precisely by saying that there is no program shorter than the sequence itself which can be used to generate the sequence, as discussed in more detail on page 1067.

▪ **Page 317 · Randomness in *Mathematica*.** *SeedRandom[n]* is the function that sets up the initial conditions for the cellular automaton. The idea of using this kind of system in general and this system in particular as a source of randomness was described in my 1987 U.S. patent number 4,691,291.

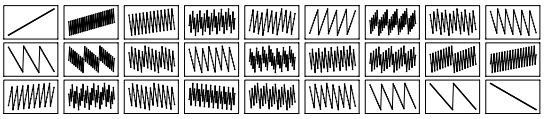
▪ **Page 321 · Cellular automata.** From the discussion here it should not be thought that in general there is necessarily anything better about generating randomness with cellular automata than with systems based on numbers. But the point is that the specific method used for making practical linear congruential generators does not yield particularly good randomness and has led to some incorrect intuition about the generation of randomness. If one goes beyond the specifics of linear congruential generators, then one can find many features of systems based on numbers that seem to be

perfectly random, as discussed in Chapter 4. In addition, one should recognize that while the complete evolution of the cellular automaton may effectively generate perfect randomness, there may be deviations from randomness introduced when one constructs a practical random number generator with a limited number of cells. Nevertheless, no such deviations have so far been found except when one looks at sequences whose lengths are close to the repetition period. (See however page 603.)

▪ **Page 321 · Card shuffling.** Another rather poor example of intrinsic randomness generation is perfect card shuffling. In a typical case, one splits the deck of cards in two, then carefully riffles the cards so as to make alternate cards come from each part of the deck. Surprisingly enough, this simple procedure, which can be represented by the function

```
s[list_] := Flatten[
    Transpose[Reverse[Partition[list, Length[list]/2]]]]
```

with or without the *Reverse*, is able to produce orderings which at least in some respects seem quite random. But by doing *Nest[s, Range[52], 26]* one ends up with a simple reversal of the original deck, as in the pictures below.



▪ **Random number generators.** A fairly small number of different types of random number generators have been used in practice, so it is possible to describe all the major ones here.

Linear congruential generators. The original suggestion made by Derrick Lehmer in 1948 was to take a number n and at each step to replace it by *Mod[a n, m]*. Lehmer used $a = 23$ and $m = 10^8 + 1$. Most subsequent implementations have used $m = 2^j$, often with $j = 31$. Such choices are particularly convenient on computers where machine integers are represented by 32 binary digits. The behavior of the linear congruential generator depends greatly on the exact choice of a. Starting with the so-called RANDU generator used on mainframe computers in the 1960s, a common choice made was $a = 65539$. But as shown in the main text, this choice leads to embarrassingly obvious regularities. Starting in the mid-1970s, another common choice was $a = 69069$. This was also found to lead to regularities, but only in six or more dimensions. (Small values of a also lead to an excess of runs of identical digits, as mentioned on page 903.)

The repetition period for a generator with rule $n \to$ *Mod[a n, m]* is given (for a and m relatively prime) by *MultiplicativeOrder[a, m]*. If m is of the form 2^j, this implies a maximum period for any a of $m/4$, achieved when *MemberQ[{3, 5}, Mod[a, 8]]*. In general the maximum period is *CarmichaelLambda[m]*, where the value $m - 1$ can be achieved for prime m.

As illustrated in the main text, when $m = 2^j$ the right-hand base 2 digits in numbers produced by linear congruential generators repeat with short periods; a digit k positions from the right will typically repeat with period no more than 2^k. When $m = 2^j - 1$ is prime, however, even the rightmost digit repeats only with period $m - 1$ for many values of a.

More general linear congruential generators use the basic rule $n \to$ *Mod[a n + b, m]*, and in this case, $n = 0$ is no longer special, and a repetition period of exactly m can be achieved with appropriate choices of a, b and m. Note that if the period is equal to its absolute maximum of m, then every possible n is always visited, whatever n one starts from. Page 962 showed diagrams that represent the evolution for all possible starting values of n.

Each point in the 2D plots in the main text has coordinates of the form *{n[i], n[i + 1]}* where *n[i + 1] = Mod[a n[i], m]*. If one could ignore the *Mod*, then the coordinates would simply be *{n[i], a n[i]}*, so the points would lie on a single straight line with slope a. But the presence of the *Mod* takes the points off this line whenever $a\,n[i] \geq m$. Nevertheless, if a is small, there are long runs of *n[i]* for which the *Mod* is never important. And that is why in the case $a = 3$ the points in the plot fall on obvious lines.

In the case $a = 65539$, the points lie on planes in 3D. The reason for this is that

```
n[i + 2] == Mod[65539^2 n[i], 2^31] ==
  Mod[6 n[i + 1] - 9 n[i], 2^31]
```

so that in computing *n[i + 2]* from *n[i + 1]* and *n[i]* only small coefficients are involved.

It is a general result related to finding short vectors in lattices that for some d the quantity *n[i + d]* can always be written in terms of the *n[i + k]*'s; $k < d$ using only small coefficients. And as a consequence, the points produced by any linear congruential generator must lie on regular hyperplanes in some number of dimensions.

(For cryptanalysis of linear congruential generators see page 1089.)

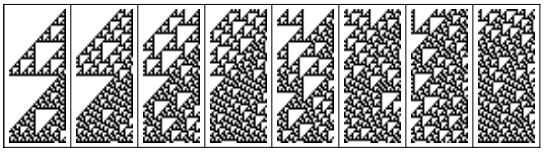
Linear feedback shift registers. Used since the 1950s, particularly in special-purpose electronic devices, these systems are effectively based on running additive cellular automata such as rule 60 in registers with a limited number

of cells and with a certain type of spiral boundary conditions. In a typical case, each cell is updated using

```
LFSRStep[list_] :=
  Append[Rest[list], Mod[list[[1]] + list[[2]], 2]]
```

with a step of cellular automaton evolution corresponding to the result of updating all cells in the register. As with additive cellular automata, the behavior obtained depends greatly on the length n of the register. The maximal repetition period of $2^n - 1$ can be achieved only if `Factor[1 + x + x^n, Modulus → 2]` finds no factors. (For $n < 512$, this is true when $n = 1, 2, 3, 4, 6, 7, 9, 15, 22, 28, 30, 46, 60, 63, 127, 153, 172, 303$ or 471. Maximal period is assured when in addition `PrimeQ[2^n - 1]`.) The pictures below show the evolution obtained for $n = 30$ with

```
NestList[Nest[LFSRStep, #, n] &,
  Append[Table[0, {n - 1}], 1], t]
```



Like additive cellular automata as discussed on page 951, states in a linear feedback shift register can be represented by a polynomial `FromDigits[list, x]`. Starting from a single 1, the state after t steps is then given by

```
PolynomialMod[x^t, {1 + x + x^n, 2}]
```

This result illustrates the analogy with linear congruential generators. And if the distribution of points generated is studied with the Cantor set geometry, the same kind of problems occur as in the linear congruential case (compare page 1094).

In general, linear feedback shift registers can have "taps" at any list of positions on the register, so that their evolution is given by

```
LFSRStep[taps_List, list_] :=
  Append[Rest[list], Mod[Apply[Plus, list[[taps]]], 2]]
```

(With taps specified by the positions of 1's in a vector of 0's, the inside of the `Mod` can be replaced by `vec . list` as on page 1087.) For a register of size n the maximal period of $2^n - 1$ is obtained whenever $x^n + $ `Apply[Plus, x^(taps-1)]` is one of the `EulerPhi[2^n - 1]/n` primitive polynomials that appear in `Factor[Cyclotomic[2^n - 1, x], Modulus → 2]`. (See pages 963 and 1084.)

One can also consider nonlinear feedback shift registers, as discussed on page 1088.

Generalized Fibonacci generators. It was suggested in the late 1950s that the Fibonacci sequence `f[n_] := f[n - 1] + f[n - 2]` modulo 2^k might be used with different choices of `f[0]` and `f[1]` as a random number generator (see page 891). This particular idea did not work well, but generalizations based on the recurrence `f[n_] := Mod[f[n - p] + f[n - q], 2^k]` have been studied extensively, for example with $p = 24$, $q = 55$. Such generators are directly related to linear feedback shift registers, since with a list of length q, each step is simply

```
Append[Rest[list], Mod[list[[1]] + list[[q - p + 1]], 2^k]]
```

Cryptographic generators. As discussed on page 598, so-called stream cipher cryptographic systems work essentially by generating a repeatable random sequence. Practical stream cipher systems can thus be used as random number generators. Starting in the 1980s, the most common example has been the Data Encryption Standard (DES) introduced by the U.S. government (see page 1085). Unless special-purpose hardware is used, however, this method has not usually been efficient enough for practical random number generation applications.

Quadratic congruential generators. Several generalizations of linear congruential generators have been considered in which nonlinear functions of n are used at each step. In fact, the first known generator for digital computers was John von Neumann's "middle square method"

```
n → FromDigits[Take[IntegerDigits[n^2, 10, 20], {5, 15}], 10]
```

In practice this generator has too short a repetition period to be useful. But in the early 1980s studies of public key cryptographic systems based on number theoretical problems led to some reinvestigation of quadratic congruential generators. The simplest example uses the rule

```
n → Mod[n^2, m]
```

It was shown that for $m = p\,q$ with p and q prime the sequence `Mod[n, 2]` was in a sense as difficult to predict as the number m is to factor (see page 1090). But in practice, the period of the generator in such cases is usually too short to be useful. In addition, there has been the practical problem that if n is stored on a computer as a 32-bit number, then n^2 can be 64 bits long, and so cannot be stored in the same way. In general, the period divides `CarmichaelLambda[CarmichaelLambda[m]]`. When m is a prime, this implies that the period can then be as long as $(m - 3)/2$. The largest m less than 2^{16} for which this is true is 65063, and the sequence generated in this case appears to be fairly random.

Cellular automaton generators. I invented the rule 30 cellular automaton random number generator in 1985. Since that time the generator has become quite widely used for a variety of applications. Essentially all the other generators discussed here have certain linearity properties which

allow for fairly complete analysis using traditional mathematical methods. Rule 30 has no such properties. Empirical studies, however, suggest that the repetition period, for example, is about $2^{0.63\,n}$, where n is the number of cells (see page 260). Note that rule 45 can be used as an alternative to rule 30. It has a somewhat longer period, but does not mix up nearby initial conditions as quickly as rule 30. (See also page 603.)

■ **Unequal probabilities.** Given a sequence a of n equally probable 0's and 1's, the following generates a single 0 or 1 with probabilities approximating $\{1-p, p\}$ to n digits:

```
Fold[{BitAnd, BitOr}[[1 + First[#2]]][#1, Last[#2]] &, 0,
  Reverse[Transpose[{First[RealDigits[p, 2, n, -1]], a}]]]
```

This can be generalized to allow a whole sequence to be generated with as little as an average of two input digits being used for each output digit.

■ **Page 323 · Sources of repeatable randomness.** In using repeatability to test for intrinsic randomness generation, one must avoid systems in which there is essentially some kind of static randomness in the environment. Sources of this include the profile of a rough solid surface, or the detailed patterns of grains inside a solid.

■ **Page 324 · Probabilistic rules.** There appears to be a discrete transition as a function of the size of the perturbations, similar to phase transitions seen in the phenomenon of directed percolation. Note that if one just uses the original cellular automata rules, then with any nonzero probability of reversing the colors of cells, the patterns will be essentially destroyed. With more complicated cellular automaton rules, one can get behavior closer to the continuous cellular automata shown here. (See also page 591.)

■ **Page 325 · Noisy cellular automata.** In correspondence with electronics, the continuous cellular automata used here can be thought of as analog models for digital cellular automata. The specific form of the continuous generalization of the modulo 2 function used is

```
λ[x_] := Exp[-10 (x - 1)^2] + Exp[-10 (x - 3)^2]
```

Each cell in the system is then updated according to $\lambda[a + c]$ for rule 90, and $\lambda[a + b + c + b\,c]$ for rule 30. Perturbations of size δ are then added using $v + \text{Sign}[v - 1/2]\,\text{Random}[]\,\delta$.

Note that the basic approach used here can be extended to allow discrete cellular automata to be approximated by partial differential equations where not only color but also space and time are continuous. (Compare page 464.)

■ **Page 326 · Repeatably random experiments.** Over the years, I have asked many experimental scientists about repeatability in seemingly random data, and in almost all cases they have told me that they have never looked for such a thing. But in a few cases they say that in fact on thinking about it they remember various forms of repeatability.

Examples where I have seen evidence of repeatable randomness as a function of time in published experimental data include temperature differences in thermal convection in closed cells of liquid helium, reaction rates in oxidation of carbon monoxide on catalytic surfaces, and output voltages from firings of excited single nerve cells. Typically there are quite long periods of time where the behavior is rather accurately repeatable—even though it may wiggle tens or hundreds in a seemingly random way—interspersed with jumps of some kind. In most cases the only credible models seem to be ones based on intrinsic randomness generation. But insofar as there is any definite model, it is inevitable that looking in sufficient detail at sufficiently many components of the system will reveal regularities associated with the underlying mechanism.

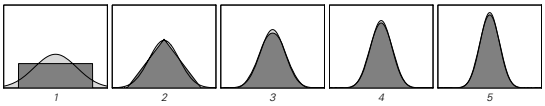
The Phenomenon of Continuity

■ **Discreteness in computer programs.** The reason for discreteness in computer programs is that the only real way we know how to construct such programs is using discrete logical structures. The data that is manipulated by programs can be continuous, as can the elements of their rules. But at some level one always gives discrete symbolic descriptions of the logical structure of programs. And it is then certainly more consistent to make both data and programs involve only discrete elements. In Chapter 12 I will argue that this approach is not only convenient, but also necessary if we are to represent our computations using processes that can actually occur in nature.

■ **Central Limit Theorem.** Averages of large collections of random numbers tend to follow a Gaussian or normal distribution in which the probability of getting value x is

```
Exp[-(x - μ)^2/(2 σ^2)]/(Sqrt[2 π] σ)
```

The mean μ and standard deviation σ are determined by properties of the random numbers, but the form of the distribution is always the same. The only conditions are that the random numbers should be statistically independent, and that their distribution should have bounded variance, so that, for example, the probability for very large numbers is rapidly damped. (The limit of an infinite collection of numbers gives $\sigma \to 0$ in accordance with the law of large numbers.) The pictures at the top of the next page show how averages of successively larger collections of uniformly distributed numbers converge to a Gaussian distribution.



The Central Limit Theorem leads to a self-similarity property for the Gaussian distribution: if one takes n numbers that follow Gaussian distributions, then their average should also follow a Gaussian distribution, though with a standard deviation that is $1/\sqrt{n}$ times smaller.

▪ **History.** That averages of random numbers follow bell-shaped distributions was known in the late 1600s. The formula for the Gaussian distribution was derived by Abraham de Moivre around 1733 in connection with theoretical studies of gambling. In the late 1700s Pierre-Simon Laplace did this again to predict the distribution of comet orbits, and showed that the same results would be obtained for other underlying distributions. Carl Friedrich Gauss made connections to the distribution of observational errors, and the relevance of the Gaussian distribution to biological and social systems was noted. Progressively more general proofs of the Central Limit Theorem were given from the early 1800s to the 1930s. Many natural systems were found to exhibit Gaussian distributions—a typical example being height distributions for humans. (Weight distributions are however closer to lognormal; compare page 1003.) And when statistical methods such as analysis of variance became established in the early 1900s it became increasingly common to assume underlying Gaussian distributions. (Gaussian distributions were also found in statistical mechanics in the late 1800s.)

▪ **Related results.** Gaussian distributions arise when large numbers of random variables get added together. If instead such variables (say probabilities) get multiplied together what arises is the lognormal distribution

```
Exp[-(Log[x] - μ)^2/(2 σ^2)]/(Sqrt[2 π] x σ)
```

For a wide range of underlying distributions the extreme values in large collections of random variables follow the Fisher-Tippett distribution

```
Exp[(x - μ)/β] Exp[-Exp[(x - μ)/β]]/β
```

related to the Weibull distribution used in reliability analysis.

For large symmetric matrices with random entries following a distribution with mean 0 and bounded variance the density of normalized eigenvalues tends to Wigner's semicircle law

```
2 Sqrt[1 - x^2] UnitStep[1 - x^2]/π
```

while the distribution of spacings between tends to

```
1/2 (π x) Exp[1/4 (-π) x^2]
```

The distribution of largest eigenvalues can often be expressed in terms of Painlevé functions.

(See also $1/f$ noise on page 969.)

▪ **Page 328 · Random walks.** In one dimension, a random walk with t steps of length 1 starting at position 0 can be generated from

```
NestList[# + (-1)^Random[Integer] &, 0, t]
```

or equivalently

```
FoldList[Plus, 0, Table[(-1)^Random[Integer], {t}]]
```

A generalization to d dimensions is then

```
FoldList[Plus, Table[0, {d}], Table[RotateLeft[PadLeft[
  {(-1)^Random[Integer]}, d], Random[Integer, d - 1]], {t}]]
```

A fundamental property of random walks is that after t steps the root mean square displacement from the starting position is proportional to $\sqrt{t}$. In general, the probability distribution for the displacement of a particle that executes a random walk is

```
With[{σ = 1}, (d/(2 π σ t))^(d/2) Exp[-d r^2/(2 σ t)]]
```

The same results are obtained, with a different value of σ, for other random microscopic rules, so long as the variance of the distribution of step lengths is bounded (as in the Central Limit Theorem).

As mentioned on page 1082, the frequency spectrum `Abs[Fourier[list]]^2` for a 1D random walk goes like $1/\omega^2$.

The character of random walks changes somewhat in different numbers of dimensions. For example, in 1D and 2D, there is probability 1 that a particle will eventually return to its starting point. But in 3D, this probability (on a simple cubic lattice) drops to about 0.341, and in d dimensions the probability falls roughly like `1/(2 d)`. After a large number of steps t, the number of distinct positions visited will be proportional to t, at least above 2 dimensions (in 2D, it is proportional to `t/Log[t]` and in 1D $\sqrt{t}$). Note that the outer boundaries of patterns like those on page 330 formed by n random walks tend to become rougher when t is much larger than `Log[n]`.

To make a random walk on a lattice with k directions in two dimensions, one can set up

```
e = Table[{Cos[2 π s/k], Sin[2 π s/k]}, {s, 0, k - 1}]
```

then use

```
FoldList[Plus, {0, 0}, Table[e[[Random[Integer, {1, k}]]], {t}]]
```

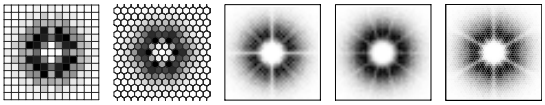
It turns out that on any regular lattice, in any number of dimensions, the average behavior of a random walk is always isotropic. As discussed in the note below, this can be viewed as a consequence of the fact that the probability distribution in a random walk depends only on

```
Sum[Outer[Times, e[[s]], e[[s]]], {s, Length[e]}]
```

and not on products of more of the `e[[s]]`.

There are nevertheless some properties of random walks that are not isotropic. The picture below, for example, shows the

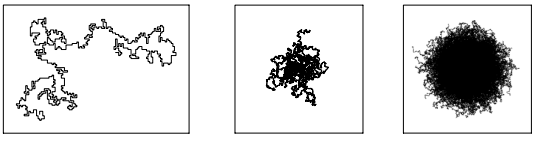
so-called extreme value distribution of positions furthest from the origin reached after 10 steps and 100 steps by random walks on various lattices.



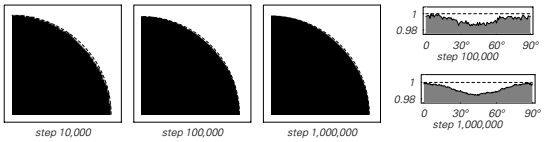
In the pictures in the main text, all particles start out at a particular position, and progressively spread out from there. But in general, one can consider sources that emit new particles every step, or absorbers and reflectors of particles. The average distribution of particles is given in general by the diffusion equation shown on page 163. The solutions to this equation are always smooth and continuous.

A physical example of an approximation to a random walk is the spreading of ink on blotting paper.

■ **Self-avoiding walks.** Any walk where the probabilities for a given step depend only on a fixed number of preceding steps gives the same kind of limiting Gaussian distribution. But imposing the constraint that a walk must always avoid anywhere it has been before (as for example in an idealized polymer molecule) leads to correlations over arbitrary times. If one adds individual steps at random then in 2D one typically gets stuck after perhaps a few tens of steps. But tricks are known for generating long self-avoiding walks by combining shorter walks or successively pivoting pieces starting with a simple line. The pictures below show some 1000-step examples. They look in many ways similar to ordinary random walks, but their limiting distribution is no longer strictly Gaussian, and their root mean square displacement after t steps varies like $t^{3/4}$. (In $d \le 4$ dimensions the exponent is close to the Flory mean field theory value $3/(2+d)$; for $d > 4$ the results are the same as without self-avoidance.)



■ **Page 331 · Basic aggregation model.** This model appears to have first been described by Murray Eden in 1961 as a way of studying biological growth, and was simulated by him on a computer for clusters up to about 32,000 cells. By the mid-1980s clusters with a billion cells had been grown, and a very surprising slight anisotropy had been observed. The pictures below show which cells occur in more than 10% of 1000 randomly grown clusters. There is a 2% or so anisotropy that appears to remain essentially fixed for clusters above perhaps a million cells, tucking them in along the diagonal directions. The width of the region of roughness on the surface of each cluster varies with the radius of the cluster approximately like $r^{1/3}$. The most extensive use of the model in practice has been for studying tumor growth: currently a typical tumor at detection contains about a billion cells, and it is important to predict what protrusions there will be that can break off and form additional tumors elsewhere.



■ **Implementation.** One way to represent a cluster is by giving a list of the coordinates at which each black cell occurs. Then starting with a single black cell at the origin, represented by *{{0, 0}}*, the cluster can be grown for t steps as follows:

```
AEvolve[t_] := Nest[AStep, {{0, 0}}, t]

AStep[c_] := (If[! MemberQ[c, #], Append[c, #],
     AStep[c]] &)[f[c] + f[{{1, 0}, {0, 1}, {-1, 0}, {0, -1}}]]
f[a_] := a[[Random[Integer, {1, Length[a]}]]]
```

This implementation can easily be extended to any type of lattice and any number of dimensions. Even with various additional optimizations, it is remarkable how much slower it is to grow a cluster with a model that requires external random input than to generate similar patterns with models such as cellular automata that intrinsically generate their own randomness.

The implementation above is a so-called type B Eden model in which one first selects a cell in the cluster, then randomly selects one of its neighbors. One gets extremely similar results with a type A Eden model in which one just randomly selects a cell from all the ones adjacent to the cluster. With a grid of cells set up in advance, each step in this type of Eden model can be achieved with

```
AStep[a_] := ReplacePart[a, 1, (#[[Random[
     Integer, {1, Length[#]}]]] &)[Position[(1 - a) Sign[
     ListConvolve[{{0, 1, 0}, {1, 0, 1}, {0, 1, 0}}, a, {2, 2}]], 1]]]
```

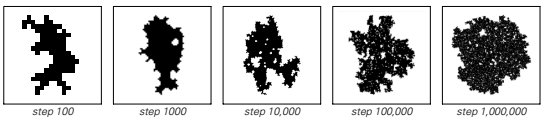
This implementation can readily be extended to generalized aggregation models (see below).

■ **Page 332 · Generalized aggregation models.** One can in general have rules in which new cells can be added only at positions whose neighborhoods match specific templates (compare page 213). There are 32 possible symmetric such rules with just 4 immediate neighbors—of which 16 lead to growth (from any seed), and all seem to yield at least

approximately circular clusters (of varying densities). Without symmetry, all sorts of shapes can be obtained, as in the pictures below. (The rule numbers here follow the scheme on page 927 with offsets *{{-1, 0}, {0, -1}, {0, 1}, {1, 0}}*). Note that even though the underlying rule involves randomness definite geometrical shapes can be produced. An extreme case is rule 2, where only a single neighborhood with a single black cell is allowed, so that growth occurs along a single line.

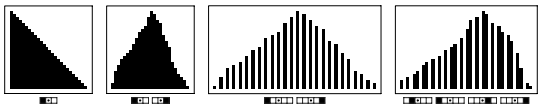
If one puts conditions on where cells can be added one can in principle get clusters where no further growth is possible. This does not seem to happen for rules that involve 4 neighbors, but with 8 neighbors there are cases in which clusters can get fairly large, but end up having no sites where further cells can be added. The pictures below show examples for a rule that allows growth except when there are exactly 1, 3 or 4 neighbors (totalistic constraint 242).

The question of what ultimate forms of behavior can occur with any sequence of random choices, starting from a given configuration with a given rule, is presumably in general undecidable. (It has some immediate relations to tiling problems and to halting problems for non-deterministic Turing machines.) With the rule illustrated above, however, those clusters that do successfully grow exhibit complicated and irregular shapes, but nevertheless eventually seem to take on a roughly circular shape, as in the pictures below.

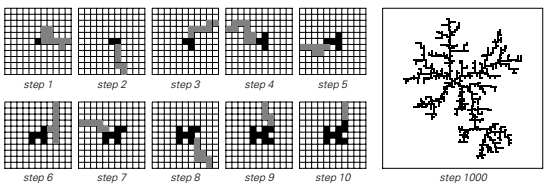


At some level the basic aggregation model of page 331 has a deterministic outcome: after sufficiently many steps every cell will be black. But most generalized aggregation models do not have this property: instead, the form of their internal patterns depends on the sequence of random choices made. Particularly with more than two colors it is however possible to arrange that the internal pattern always ends up being the same, or at least has patches that are the same—essentially by using rules with the confluence property discussed on page 1036.

The pictures below show 1D generalized aggregation systems with various templates. The second one is the analog of the system from page 331.

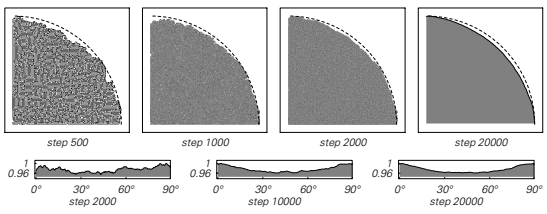


▪ **Page 333 · Diffusion-limited aggregation (DLA).** While many 2D cellular automata produce intricate nested shapes, the aggregation models shown here seem to tend to simple limiting shapes. Most likely there are some generalized aggregation models for which this is not the case. And indeed this phenomenon has been seen in other systems with randomness in their underlying rules. An example studied extensively in the 1980s is diffusion-limited aggregation (DLA). The idea of this model is to add cells to a cluster one at a time, and to determine where a cell will be added by seeing where a random walk that starts far from the cluster first lands on a square adjacent to the cluster. An example of the behavior obtained in this model is shown below:

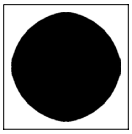


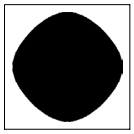
The lack of smooth overall behavior in this case can perhaps be attributed to the global probing of the cluster that is effectively done by each incoming random walk. (See also page 994.)

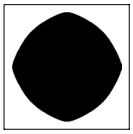
▪ **Page 334 · Code 746.** Much as in the aggregation model above, the pictures below show that there is a slight deviation from perfect circular growth, with an anisotropy that appears to remain roughly fixed at perhaps 4% above a few thousand steps (corresponding to patterns with a few million cells).



■ **Other rules.** The pictures below show patterns generated after 10,000 steps with several rules, starting respectively from rows of 7, 6, 7 and 11 cells (compare pages 177 and 181). The outer boundaries are somewhat smooth, though definitely not circular. In the second rule shown, the interior of the pattern always continues to change; in the others it remains essentially fixed.


code 746


code 29408


code 175850


code 174826

■ **Isotropy.** Any pattern grown from a single cell according to rules that do not distinguish different directions on a lattice must show the same symmetry as the lattice. But we have seen that in fact many rules actually yield almost circular patterns with much higher symmetry. One can characterize the symmetry of a pattern by taking the list v of positions of cells it contains, and looking at tensors of successive ranks n:

```
Apply[Plus,
  Map[Apply[Outer[Times, ##] &, Table[#, {n}]] &, v]]
```

For circular or spherical patterns that are perfectly isotropic in d dimensions these tensors must all be proportional to

```
(d - 2)!! Array[Apply[Times, Map[(1 - Mod[#, 2]) (# - 1)!! &,
  Table[Count[{##}, i], {i, d}]]] &, Table[d, {n}]]/(d + n - 2)!!
```

For odd n this is inevitably true for any lattice with mirror symmetry. But for even n it can fail. For a square lattice, it still nevertheless always holds up to $n = 2$ (so that the analogs of moments of inertia satisfy $\mathcal{I}_{xx} == \mathcal{I}_{yy}$, $\mathcal{I}_{xy} == \mathcal{I}_{yx} == 0$). And for a hexagonal lattice it holds up to $n = 4$. But when $n = 4$ isotropy requires the $\{1, 1, 1, 1\}$ and $\{1, 1, 2, 2\}$ tensor components to have ratio $\beta = 3$—while square symmetry allows these components to have any ratio. (In general there will be more than one component unless the representation of the lattice symmetry group carried by the rank n tensor is irreducible.) In 3D no regular lattice forces isotropy beyond $n = 2$, while in 4D the SO(8) lattice works up to $n = 4$, in 8D the E_8 lattice up to $n = 6$, and in 24D the Leech lattice up to $n = 10$. (Lattices that give dense sphere packings tend to show more isotropy.) Note that isotropy can also be characterized using analogs of multipole moments, obtained in 2D by summing $r_i \, \text{Exp}[i \, n \, \theta_i]$, and in higher dimensions by summing appropriate *SphericalHarmonicY* or *GegenbauerC* functions. For isotropy, only the $n = 0$ moment can be nonzero. On a 2D lattice with m directions, all moments are forced to be zero except when m divides n. (Sums of squares of moments of given order in general provide rotationally invariant measures of anisotropy—equal to pair correlations weighted with *LegendreP* or *GegenbauerC* functions.)

Even though it is not inevitable from lattice symmetry, one might think that if there is some kind of effective randomness in the underlying rules then sufficiently large patterns would still often show some sort of average isotropy. And at least in the case of ordinary random walks, they do, so that for example, the ratio averaged over all possible walks of $n = 4$ tensor components after t steps on a square lattice is $\beta = 3 + 2/(t - 1)$, converging to the isotropic value 3, and the ratio of $n = 6$ components is $5 - 4/(t - 1) + 32/(3\,t - 4)$. For the aggregation model of page 331, β also decreases with t, reaching 4 around $t = 10$, but now its asymptotic value is around 3.07.

In continuous systems such as partial differential equations, isotropy requires that coordinates in effect appear only in ∇. In most finite difference approximations, there is presumably isotropy in the end, but the rates of convergence are almost inevitably rather different in different directions relative to the lattice.

■ **Page 336 · Domains.** Some of the effective rules for interfaces between black and white domains are easy to state. Given a flat interface, the layer of cells immediately on either side of this interface behaves like the rule 150 1D cellular automaton. On an infinitely long interface, protrusions of cells with one color into a domain of the opposite color get progressively smaller, eventually leaving only a certain pattern of cells in the layer immediately on one side of the interface. 90° corners in an otherwise flat interface effectively act like reflective boundary conditions for the layer of cells on top of the interface.

The phenomenon of domains illustrated here is also found in various 2D cellular automata with 4-neighbor rather than 8-neighbor rules. One example is totalistic code 52, which is a direct analog in the 4-neighbor case of the rule illustrated here. Other examples are outer totalistic codes 111, 293, 295 and 920. The domain boundaries in these cases, however, are not as clear as for the 8-neighbor totalistic rule with code 976 that is shown here.

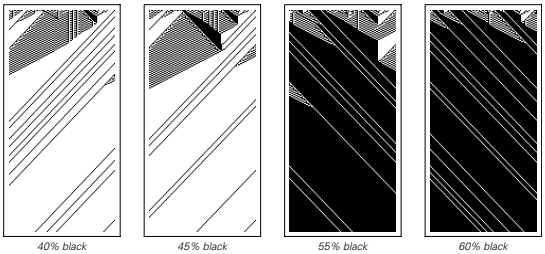
■ **Spinodal decomposition.** The separation into progressively larger black and white regions seen in the cellular automata shown here is reminiscent of the phenomena that occur for example in the separation of randomly mixed oil and water. Various continuous models of such processes have been proposed, notably the Cahn-Hilliard equation from 1958. One feature often found is that the average radius of "droplets" increases with time roughly like $t^{1/3}$.

Origins of Discreteness

■ **Page 339 · 1D transitions.** There are no examples of the phenomenon shown here among the 256 rules with two possible colors and depending only on nearest neighbors. Among the 4,294,967,296 rules that depend on next-nearest neighbors, there are a handful of examples, including rules with numbers 4196304428, 4262364716, 4268278316 and 4266296876. The behavior obtained with the first of these rules is shown below. An example that depends on three neighbors on each side was discovered by Peter Gacs, Georgii Kurdyumov and Leonid Levin in 1978, following work on how reliable electronic circuits can be built from unreliable components by Andrei Toom:

```
{a1_, a2_, a3_, a4_, a5_, a6_, a7_} →
  If[If[a4 == 1, a1 + a3 + a4, a4 + a5 + a7] ≥ 2, 1, 0]
```

The 4-color rule shown in the text is probably the clearest example available in one dimension. It has rule number 2948697645239957498148900977948124938 24.



■ **Page 340 · 2D transitions.** The simplest symmetrical rules (such as 4-neighbor totalistic code 56) which make the new color of a cell be the same as the majority of the cells in its neighborhood do not exhibit the discrete transition phenomenon, but instead lead to fixed regions of black and white. The 4-neighbor rule with totalistic code 52 can be used as an alternative to the second rule shown here. A probabilistic version of the first rule shown here was discussed by Andrei Toom in 1980.

■ **Phase transitions.** The discrete transitions shown in cellular automata in this section are examples of general phenomena known in physics as phase transitions. A phase transition can be defined as any discontinuous change that occurs in a system with a large number of components when a parameter associated with that system is varied. (Some physicists might argue for a somewhat narrower definition that allows only discontinuities in the so-called partition function of equilibrium statistical mechanics, but for many of the most interesting applications, the definition I use is the appropriate one.) Standard examples of phase transitions include boiling, melting, sublimation (solids such as dry ice turning into gases), loss of magnetization when a ferromagnet is heated, alignment of molecules in liquid crystals above a certain electric field (the basis for liquid crystal displays), and the onset of superconductivity and superfluidity at low temperatures.

It is conventional to distinguish two kinds of phase transitions, often called first-order and higher-order. First-order transitions occur when a system has two possible states, such as liquid and gas, and as a parameter is varied, which of these states is the stable one changes. Boiling and melting are both examples of first-order transitions, as is the phenomenon shown in the cellular automaton in the main text. Note that one feature of first-order transitions is that as soon as the transition is passed, the whole system always switches completely from one state to the other.

Higher-order transitions are in a sense more gradual. On one side of the transition, a system is typically completely disordered. But when the transition is passed, the system does not immediately become completely ordered. Instead, its order increases gradually from zero as the parameter is varied. Typically the presence of order is signalled by the breaking of some kind of symmetry—say of rotational symmetry by the spontaneous selection of a preferred direction.

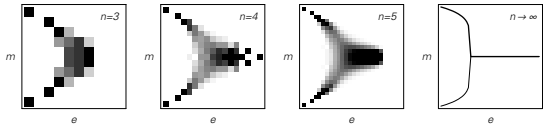
■ **The Ising model.** The 2D Ising model is a prototypical example of a system with a higher-order phase transition. Introduced by Wilhelm Lenz in 1920 as an idealization of ferromagnetic materials (and studied by Ernst Ising) it involves a square array *s* of spins, each either up or down (+1 or -1), corresponding to two orientations for magnetic moments of atoms. The magnetic energy of the system is taken to be

```
e[s_] := -1/2 Apply[Plus, s ListConvolve[
          {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}}, s, 2], {0, 1}]
```

so that each pair of adjacent spins contributes -1 when they are parallel and +1 when they are not. The overall magnetization of the system is given by `m[s_] := Apply[Plus, s, {0, 1}]`.

In physical ferromagnetic materials what is observed is that at high temperature, corresponding to high internal energy, there is no overall magnetization. But when the temperature goes below a critical value, spins tend to line up, and an overall magnetization spontaneously develops. In the context of the 2D Ising model this phenomenon is associated with the fact that those configurations of a large array of spins that have high total energy are overwhelmingly likely to have near zero overall magnetization, while those that have low

total energy are overwhelmingly likely to have nonzero overall magnetization. For an $n \times n$ array s of spins there are a total of 2^{n^2} possible configurations. The pictures below show the results of picking all configurations with a given energy $e[s]$ (cyclic boundary conditions are assumed) and then working out their distribution of magnetization values $m[s]$. Even for small n the pictures demonstrate that for large $e[s]$ the magnetization $m[s]$ is likely to be close to zero, but for smaller $e[s]$ two branches approaching +1 and -1 appear. In the limit $n \to \infty$ the distribution of magnetization values becomes sharp, and a definite discontinuous phase transition is observed.



Following the work of Lars Onsager around 1944, it turns out that an exact solution in terms of traditional mathematical functions can be found in this case. (This seems to be true only in 2D, and not in 3D or higher.) Almost all spin configurations with $e[s] > -\sqrt{2}$ (where here and below all quantities are divided by the total number of spins, so that $-2 \le e[s] \le 2$ and $-1 \le m[s] \le +1$) yield $m[s] == 0$. But for smaller $e[s]$ one can show that

```
Abs[m[s]] == (1 - Sinh[2 β]^-4)^1/8
```

where β can be deduced from

```
e[s] == -(Coth[2 β] (1 + 2 EllipticK[4 Sech[2 β]^2 Tanh[2 β]^2] (-1 + 2 Tanh[2 β]^2))/π))
```

This implies that just below the critical point $e_0 = -\sqrt{2}$ (which corresponds to $\beta = Log[1 + \sqrt{2}]/2$) $Abs[m] \sim (e_0 - e)^{1/8}$, where here 1/8 is a so-called critical exponent. (Another analytical result is that for $e \sim e_0$ correlations between pairs of spins can be expressed in terms of Painlevé functions.)

Despite its directness, the approach above of considering sets of configurations with specific energies $e[s]$ is not how the Ising model has usually been studied. Instead, what has normally been done is to take the array of spins to be in thermal equilibrium with a heat bath, so that, following standard statistical mechanics, each possible spin configuration occurs with probability $Exp[-\beta e[s]]$, where β is inverse temperature. It nevertheless turns out that in the limit $n \to \infty$ this so-called canonical ensemble approach yields the same results for most quantities as the microcanonical approach that I have used; β simply appears as a parameter, as in the formulas above.

About actual spin systems evolving in time the Ising model itself does not make any statement. But whenever the evolution is ergodic, so that all states of a given energy are visited with equal frequency, the average behavior obtained will at least eventually correspond to the average over all states discussed above.

In Monte Carlo studies of the Ising model one normally tries to sample states with appropriate probabilities by randomly flipping spins according to a procedure that can be thought of as emulating interaction with a heat bath. But in most actual physical spin systems it seems unlikely that there will be so much continual interaction with the environment. And from my discussion of intrinsic randomness generation it should come as no surprise that even a completely deterministic rule for the evolution of spins can make the system visit possible states in an effectively random way.

Among the simplest possible types of rules all those that conserve the energy $e[s]$ turn out to have behavior that is too simple and regular. And indeed, of the 4096 symmetric 5-neighbor rules, only identity and complement conserve $e[s]$. Of the 2^{32} general 5-neighbor rules 34 conserve $e[s]$—but all have only very simple behavior. (Compositions of several such rules can nevertheless yield complex behavior. Note that as indicated on page 1022, 34 of the 256 elementary 1D rules conserve the analog of $e[s]$.) Of the 262,144 9-neighbor outer totalistic rules the only ones that conserve $e[s]$ are identity and complement. But among all 2^{512} 9-neighbor rules, there are undoubtedly examples that show effectively random behavior. One marginally more complicated case effectively involving 13 neighbors is

```
IsingEvolve[list_, t_Integer] :=
  First[Nest[IsingStep, {list, Mask[list]}, t]]
IsingStep[{a_, mask_}] := {MapThread[
    If[#2 == 2 && #3 == 1, 1 - #1, #1] &, {a, ListConvolve[
      {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}}, a, 2], mask}, 2], 1 - mask}
```

where

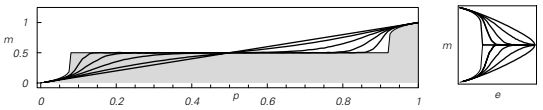
```
Mask[list_] := Array[Mod[#1 + #2, 2] &, Dimensions[list]]
```

is set up so that alternating checkerboards of cells are updated on successive steps.

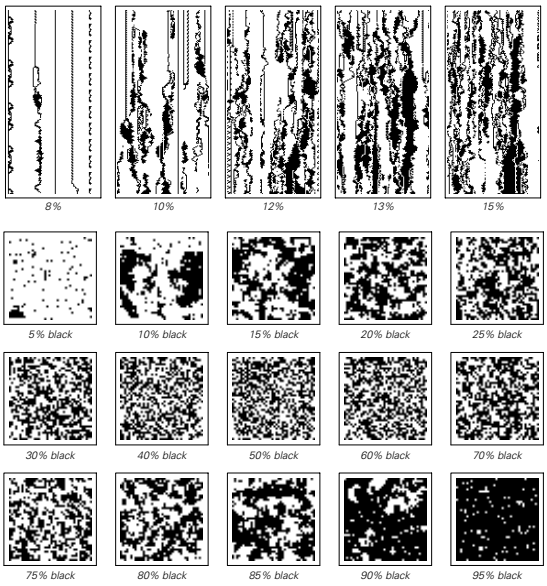
One can see a phase transition in this system by looking at the dependence of behavior on conserved total energy $e[s]$. If there are no correlations between spins, and a fraction p of them are +1, then $m[s] == p$ and $e[s] == -2(1 - 2p)^2$. And since the evolution conserves $e[s]$ changing the initial value of p allows one to sample different total energies. But since the evolution does not conserve $m[s]$ the average of this after many steps can be expected to be typical of all possible states of given $e[s]$.

The pictures at the top of the next page show the values of $m[s]$ (densities of +1 cells) after 0, 10, 100 and 1000 steps for a 500×500 system as a function of the initial values of $m[s]$ and $e[s]$. Also shown is the result expected for an infinite system at infinite time. (The slow approach to this limit can

be viewed as being a consequence of smallness of finite size scaling exponents in Ising-like systems.)



The phase transition in the Ising model is associated with a lack of smoothness in the dependence of the final m value on e or the initial value p of m in limiting cases of the pictures above. The transition occurs at $e = -\sqrt{2}$, corresponding to $p = (1 \pm 2^{-1/4})/2$. The pictures show typical configurations generated after 1000 steps from various initial densities, as well as slices through their evolution.



And what one sees at least roughly is that right around the phase transition there are patches of black and white of all sizes, forming an approximately nested random pattern. (See also pages 989 and 1149.)

▪ **General features of phase transitions.** To reproduce the Ising model, a cellular automaton must have several special properties. In addition to conserving energy, its evolution must be reversible in the sense discussed on page 435. And with the constraint of reversibility, it turns out that it is impossible to get a non-trivial phase transition in any 1D system with the kind of short-range interactions that exist in a cellular automaton. But in systems whose evolution is not reversible, it is possible for phase transitions to occur in 1D, as the examples in the main text show.

One point to notice is that the sharp change which characterizes any phase transition can only be a true discontinuity in the limit of an infinitely large system. In the case of the system on page 339, for example, it is possible to find special configurations with a finite total number of cells which lead to behavior opposite to what one expects purely on the basis of their initial density of black cells. When the total number of cells increases, however, the fraction of such configurations rapidly decreases, and in the infinite size limit, there are no such configurations, and a truly discontinuous transition occurs exactly at density 1/2.

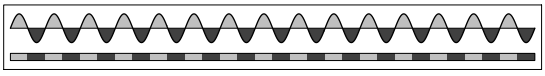
The discrete nature of phase transitions was at one time often explained as a consequence of changes in the symmetry of a system. The idea is that symmetry is either present or absent, and there is no continuous variation of level of symmetry possible. Thus, for example, above the transition, the Ising model treats up and down spins exactly the same. But below the transition, it effectively makes a choice of one spin direction or the other. Similarly, when a liquid freezes into a crystalline solid, it effectively makes a choice about the alignment of the crystal in space. But in boiling, as well as in a number of model examples, there is no obvious change of symmetry. And from studying phase transitions in cellular automata, it does not seem that an interpretation in terms of symmetry is particularly useful.

A common feature of phase transitions is that right at the transition point, there is competition between both phases, and some kind of nested structure is typically formed, as discussed on page 273 and above. The overall form and fractal dimension of this nested structure is typically independent of small-scale features of the system, making it fairly universal, and amenable to analysis using the renormalization group approach (see page 955).

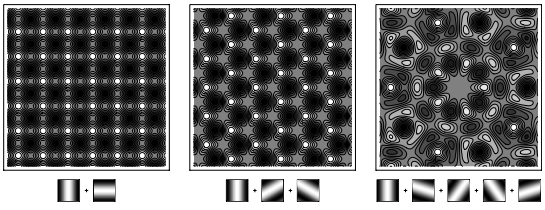
▪ **Percolation.** A simple example of a phase transition studied extensively since the 1950s involves taking a square lattice and filling in at random a certain density of black cells. In the limit of infinite size, there is a discrete transition at a density of about 0.592746, with zero probability below the transition to find a connected "percolating" cluster of black cells spanning the lattice, and unit probability above. (For a triangular lattice the critical density is exactly 1/2.) One can also study directed percolation in which one takes account of the connectivity of cells only in one direction on the lattice. (Compare the probabilistic cellular automata on pages 325 and 591. Note that the evolution of such systems is also analogous to the process of applying transfer matrices in studies of spin systems like Ising models.)

■ **Page 341 · Rate equations.** In standard chemical kinetics one assumes that molecules are uniformly distributed in space, so that the rates for particular reactions are proportional to the products of the densities of the molecules that react in them. Conditions for equilibrium where rates balance thus tend to be polynomial equations for densities—with discontinuous jumps in solutions sometimes occurring as parameters are changed. Analogous equations arise in probabilistic approximations to systems like cellular automata, as on page 953. But here—as well as in fast chemical reactions—correlations in spatial arrangements of elements tend to be important, invalidating simple probabilistic approaches. (For the cellular automaton on page 339 the simple condition for equilibrium is $p == p^2 (3 - 2 p)$, which correctly implies that 0, 1/2 and 1 are possible equilibrium densities.)

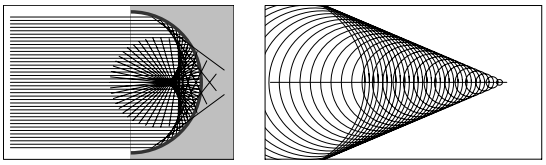
■ **Discreteness in space.** Many systems with continuous underlying rules generate discrete cellular structures in space. One common mechanism is for a wave of a definite wavelength to form (see page 988), and then for some feature of each cycle of this wave to be picked out, as in the picture below. In Chladni figures of sand on vibrating plates and in cloud streets in the atmosphere what happens is that material collects at points of zero displacement. And when a stream of water breaks up into discrete drops what happens is that oscillation minima yield necks that break.



Superpositions of waves at different angles can lead to various 2D cellular structures, as in the pictures below (compare page 1078).



Various forms of focusing and accumulation can also lead to discreteness in continuous systems. The first picture below shows a caustic or catastrophe in which a continuous distribution of light rays are focused by a circular reflector onto a discrete line with a cusp. The second picture shows a shock wave produced by an accumulation of circular waves emanating from a moving object—as seen in wakes of ships, sonic booms from supersonic aircraft, and Cerenkov light from fast-moving charged particles.



The Problem of Satisfying Constraints

■ **Rules versus constraints.** See page 940.

■ **NP completeness.** Finding 2D patterns that satisfy the constraints in the previous section is in general a so-called NP-complete problem. And this means that no known algorithm can be expected to solve this problem exactly for a size n array (say with given boundaries) in much less than 2^n steps (see page 1145). The same is true even if one allows a small fraction of squares to violate the constraints. However, the 1D version of the problem is not NP-complete, and in fact there is a specific rather efficient algorithm described on page 954 for solving it. Nevertheless, the procedures discussed in this section do not manage to make use of such specific algorithms, and in fact typically show little difference between problems that are and are not formally NP-complete.

■ **Page 343 · Distribution.** The distribution shown here rapidly approaches a Gaussian. (Note that in a 5×5 array, there are 10 interior squares that are subject to the constraints, while in a 10×10 array there are 65.) Very similar results seem to be obtained for constraints in a wide range of discrete systems.

■ **Page 346 · Implementation.** The number of squares violating the constraint used here is given by

```
Cost[list_] := Apply[Plus, Abs[list - RotateLeft[list]]]
```

When applied to all possible patterns, this function yields a distribution with Gaussian tails, but with a sharp point in the middle. Successive steps in the iterative procedure used on this page are given by

```
Move[list_] := (If[Cost[#] < Cost[list], #, list] &)[
    MapAt[1 - # &, list, Random[Integer, {1, Length[list]}]]]
```

while those in the procedure on page 347 have $\leq$ in place of $<$. The third curve shown on page 346 is obtained from

```
Table[Cost[IntegerDigits[i, 2, n]], {i, 0, 2^n - 1}]
```

There is no single ordering that makes all states which can be reached by changing a single square be adjacent. However, the ordering defined by *GrayCode* from page 901 does do this for one particular sequence of single square changes. The resulting curve is very similar to what is already shown.

■ **Page 347 · Iterative improvement.** The borders of the regions of black and white in the picture shown here essentially

follow random walks and annihilate in pairs so that their number decreases with time like $1/\sqrt{t}$. In 2D the regions are more complicated and there is no such simple behavior. Indeed starting from a particular state it is for example not clear whether it is ever possible to reach all other states.

■ **Gradient descent.** A standard method for finding a minimum in a smooth function `f[x]` is to use

```
FixedPoint[# - a f'[#] &, x0]
```

If there are local minima, then which one is reached will depend on the starting point x_0. It will not necessarily be the one closest to x_0 because of potentially complicated overshooting effects associated with the step size a. Newton's method for finding zeros of `f[x]` is related and is given by

```
FixedPoint[# - f[#]/f'[#] &, x0]
```

■ **Combinatorial optimization.** The problem of coming as close as possible to satisfying constraints in an arrangement of black and white squares is a simple example of a combinatorial optimization problem. In general, such problems involve minimization of a quantity that is determined by the arrangement of some set of discrete elements. A typical example is finding a placement of components in a 2D circuit so that the total length of wire necessary to the connect these components is minimized (related to the so-called travelling salesman problem). In using iterative procedures to solve combinatorial optimization problems, one issue is what kind of changes should be made at each step. In the main text we considered changing just one square at a time. But one can also change larger numbers of squares, or, for example, interchange whole blocks of squares. In general, the larger the changes made, the faster one can potentially approach a minimum, but the greater the chance is of overshooting. In the main text, we assumed that at each step we should always move closer to the minimum, or at least not get further away. But in trying to get over the kind of bumps shown in the third curve on page 346 it is sometimes better also to allow some probability of moving away from the minimum at a particular step. One approach is simulated annealing, in which one starts with this probability being large, and progressively decreases it. The notion is that at the beginning, one wants to move easily over the coarse features of a jagged curve, but then later home in on details. If the curve has a nested form, which appears to be the case in some combinatorial optimization problems, then this scheme can be expected to be at least somewhat effective. For the problems considered in the main text, simulated annealing provides some improvement but not much.

■ **Biologically motivated schemes.** The process of biological evolution by natural selection can be thought of as an iterative procedure for optimization. Usually, however, what is being optimized is some aspect of the form or behavior of an organism, which represents a very complicated constraint on the underlying genetic material. (It is as if one is defining constraints on the initial conditions for a cellular automaton by looking at the pattern generated by the cellular automaton after a long time.) But the strategies of biological evolution can also be used in trying to satisfy simpler constraints. Two of the most important strategies are maintaining a whole population of individuals, not just the single best result so far, and using sex to produce large-scale mixing. But once again, while these strategies may in some cases lead to greater efficiency, they do not usually lead to qualitative differences. (See also page 1105.)

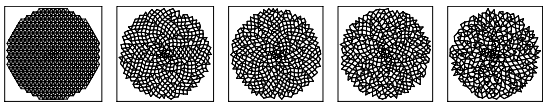
■ **History.** Work on combinatorial optimization started in earnest in the late 1950s, but by the time NP completeness was discovered in 1971 (see page 1143) it had become clear that finding exact solutions would be very difficult. Approximate methods tended to be constructed for specific problems. But in the early 1980s, simulated annealing was suggested by Scott Kirkpatrick and others as one of the first potentially general approaches. And starting in the mid-1980s, extensive work was done on biologically motivated so-called genetic algorithms, which had been advocated by John Holland since the 1960s. Progress in combinatorial optimization is however often difficult to recognize, because there are almost no general results, and results that are quoted are often sensitive to details of the problems studied and the computer implementations used.

■ **Page 349 · 2D cellular automata.** The rule numbers are specified as on page 927.

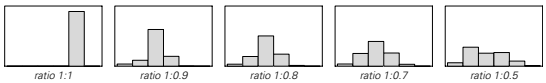
■ **Page 349 · Circle packings.** Hexagonal packing of equal circles has been known since early antiquity (e.g. the fourth picture on page 43). It fills a fraction $\pi/\sqrt{12} \simeq 0.91$ of area—which was proved maximal for periodic packings by Carl Friedrich Gauss in 1831 and for any packing by Axel Thue in 1910 and László Fejes Tóth in 1940. Much has been done to study densest packings of limited numbers of circles into various shapes, as well as onto surfaces of spheres (as in golf balls, pollen grains or radiolarians). Typically it has been found that with enough circles, patches of hexagonal packing always tend to form. (See page 987.)

For circles of unequal sizes rather little has been done. A procedure analogous to the one on page 350 was introduced by Charles Bennett in 1971 for 3D spheres (relevant for binary alloys). The picture below shows the

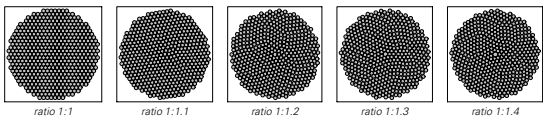
network of contacts between circles in the cases from page 350. Note that with the procedure used, each new circle added must immediately touch two existing ones, though subsequently it may get touched by varying numbers of other circles.



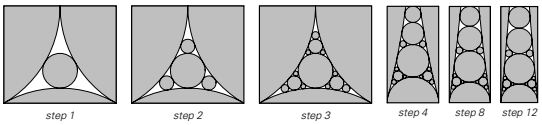
The distribution of numbers of circles that touch a given circle changes with the ratio of circle sizes, as in the picture below. The total filling fraction seems to vary fairly smoothly with this ratio, though I would not be surprised if some small-scale jumps were present.



Note that even a single circle of different size in the center can have a large-scale effect on the results of the procedure, as illustrated in the pictures below.



Finding densest packings of n circles is in general like solving quadratic programming problems with about n^2 constraints. But at least for many size ratios I suspect that the final result will simply involve each kind of circle forming a separated hexagonally-packed region. This will not happen, however, for size ratios $\leq 2/\sqrt{3} - 1 \simeq 0.15$, since then the small circles can fit into the interstices of an ordinary hexagonal pattern, yielding a filling fraction $1/18\,(17\sqrt{13} - 24)\,\pi \simeq 0.95$. The picture below shows what happens if one repeatedly inserts circles to form a so-called Apollonian packing derived from the problem studied by Apollonius of finding a circle that touches three others. At step t, 3^{t-1} circles are added for each original circle, and the network of tangencies among circles is exactly example (a) from page 509. Most of the circles added at a given step are not the same size, however, making the overall geometry not straightforwardly nested. (The total numbers of different sizes of circles for the first few steps are $\{2, 3, 5, 10, 24, 63, 178, 521\}$. At step 3, for example, the new circles have radii $(25 - 12\sqrt{3})/193$ and $(19 - 6\sqrt{3})/253$. In general, the radius of a circle inscribed between three other touching circles that have radii p, q, r is $p\,q\,r/(p\,q + p\,r + q\,r + 2\,\mathrm{Sqrt}[p\,q\,r\,(p + q + r)])$.) In the limit of an infinite number of steps the filling fraction tends to 1, while the region left unfilled has a fractal dimension of about 1.3057.



To achieve filling fraction 1 requires arbitrarily small circles, but there are many different arrangements of circles that will work, some not even close to nested. When actual granular materials are formed by crushing, there is probably some tendency to generate smaller pieces by following essentially substitution system rules, and the result may be a nested distribution of sizes that allows an Apollonian-like packing.

Apollonian packings turn out to correspond to limit sets invariant under groups of rational transformations in the complex plane. Note that as on page 1007 packings can be constructed in which the sizes of circles vary smoothly with position according to a harmonic function.

▪ **Sphere packings.** The 3D face-centered cubic (fcc) packing shown in the main text has presumably been known since antiquity, and has been used extensively for packing fruit, cannon balls, etc. It fills space with a density $\pi/\sqrt{18} \simeq 0.74$, which Johannes Kepler suggested in 1609 might be the maximum possible. This was proved for periodic packings by Carl Friedrich Gauss in 1831, and for any packing by Thomas Hales in 1998. (By offsetting successive layers hexagonal close packing (hcp) can be obtained; this has the same density as fcc, but has a trapezoid-rhombic dodecahedron Voronoi diagram—see note below and page 929—rather than an ordinary rhombic dodecahedron.)

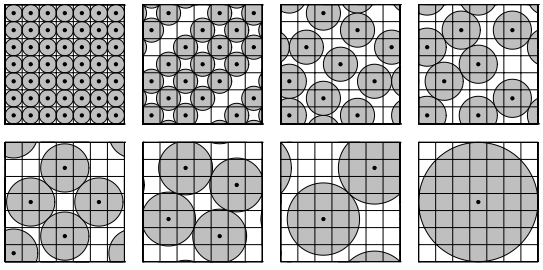
Random packings of spheres typically have densities around 0.64 (compared to 0.74 for fcc). Many of their large pores appear to be associated with poor packing of tetrahedral clusters of 4 spheres. (Note that individual such clusters—as well as for example 13-sphere approximate icosahedra—represent locally dense packings.)

It is common for shaking to cause granular materials (such as coffee or sand grains) to settle and pack at least a few percent better. Larger objects normally come to the top (as with mixed nuts, popcorn or pebbles and sand), essentially because the smaller ones more easily fall through interstices.

▪ **Higher dimensions.** In no dimension above 3 is it known for certain what configuration of spheres yields the densest packing. Cases in which spheres are arranged on repetitive lattices are related to error-correcting codes and groups. Up to 8D, the densest packings of this type are known to be ones obtained by successively adding layers individually

optimized in each dimension. And in fact up to 26D (with the exception of 11 through 13) all the densest packings known so far are lattices that work like this. In 8D and 24D these lattices are known to be ones in which each sphere touches the maximal number of others (240 and 196560 respectively). (In 8D the lattice also corresponds to the root vectors of the Lie group E_8; in 24D it is the Leech lattice derived from a Golay code, and related to the Monster Group). In various dimensions above 10 packings in which successive layers are shifted give slightly higher densities than known lattices. In all examples found so far the densest packings can always be repetitive; most can also be highly symmetrical—though in high dimensions random lattices often do not yield much worse results.

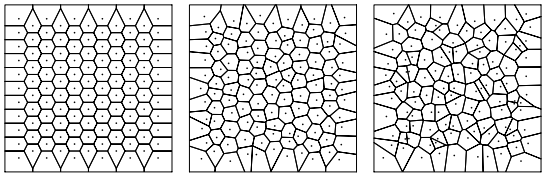
▪ **Discrete packings.** The pictures below show a discrete analog of circle packing in which one arranges as many circles as possible with a given diameter on a grid. (The grid is assumed to wrap around.)



The pictures show all the distinct maximal cases that exist for a 7×7 grid, corresponding to possible circles with diameters $Sqrt[m^2 + n^2]$. Already some of these are difficult to find. And in fact in general finding such packings is an NP-complete problem: it is equivalent to the problem of finding the maximum clique (completely connected set) in the graph whose vertices are joined whenever they correspond to grid points on which non-overlapping circles could be centered.

On large grids, optimal packings seem to approach rational approximations to hexagonal packings. But what happens if one generalizes to allow circles of different sizes is not clear.

▪ **Voronoi diagrams.** The Voronoi diagram for a set of points shows the region around each point in which one is closer to that point than to any other. (The edges of the regions are thus like watersheds.) The pictures below show a few examples. In 2D the regions in a Voronoi diagram are always polygons, and in 3D polyhedra. If all the points lie on a repetitive lattice each region will always be the same, and is often known as a Wigner-Seitz cell or a Dirichlet domain. For a simple cubic lattice the regions are cubes with 6 faces. For an fcc lattice they are rhombic dodecahedra with 12 faces and for a bcc lattice they are truncated octahedra (tetradecahedra) with 14 faces. (Compare page 929.)



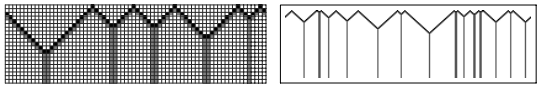
Voronoi diagrams for irregularly distributed points have found many applications. In 2D they are used in studies of animal territories, retail store utilization and municipal districting. In 3D they are used as simple models of foams, grains in solids, assemblies of biological cells and self-gravitating regions in primordial galaxy formation. Voronoi diagrams are relevant whenever there is growth in all directions at an identical speed from a collection of seed points. (In high dimensions they also appear immediately in studying error-correcting codes.)

Modern computational geometry has provided efficient algorithms for constructing Voronoi diagrams, and has allowed them to be used in mesh generation, point location, cluster analysis, machining plans and many other computational tasks.

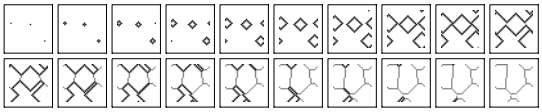
▪ **Discrete Voronoi diagrams.** The $k = 3$, $r = 1$ cellular automaton

$$\{\{0 \mid 1, n : (0 \mid 1), 0 \mid 1\} \to n, \{\_, 0, \_\} \to 2, \{\_, n\_, \_\} \to n - 1\}$$

is an example of a system that generates discrete 1D Voronoi diagrams by having regions that grow from every initial black cell, but stop whenever they meet, as shown below.



Analogous behavior can also be obtained in 2D, as shown for a 2D cellular automaton in the pictures below.



▪ **Brillouin zones.** A region in an ordinary Voronoi diagram shows where a given point is closest. One can also consider higher-order Voronoi diagrams in which each region shows where a given point is the k^{th} closest. The total area of each region is the same for every k, but some complexity in shape is seen, though for large k they always in a sense

approximate circles. 3D versions of such regions have been encountered in studies of quantum mechanical properties of crystals since the 1930s.

■ **Packing deformable objects.** If one pushes together identical deformable objects in 2D they tend to arrange themselves in a regular hexagonal array—and this configuration is known to minimize total boundary length. In 3D the arrangement one gets is typically not very regular—although as noted at various times since the 1600s individual objects often have pentagonal faces suggestive of dodecahedra. (The average number of faces for each object depends on the details of the random process used to pack them, but is typically around 14. Note that for a 3D Voronoi diagram with randomly placed points, the average number of faces for each region is $2 + 48\,\pi^2/35 \simeq 15.5$.) It was suggested by William Thomson (Kelvin) in 1887 that an array of 14-faced tetradecahedra on a bcc lattice might yield minimum total face area. But in 1993 Denis Weaire and Robert Phelan discovered a layered repetitive arrangement of 12- and 14-faced polyhedra (average 13.5) that yields 0.003 times less total area. It seems likely that there are polyhedra which fill space in a less regular way and yield still smaller total area. (Note that if the surfaces minimize area like soap films they are slightly curved in all these cases. See also pages 1007 and 1039.)

■ **Page 351 · Protein folding.** When the molecular structure of proteins was first studied in the 1950s it was assumed that given their amino acid sequences pure minimization of energy would determine their often elaborate overall shapes. But by the 1990s it was fairly clear that in fact many details of the actual processes by which proteins are assembled can greatly affect their specific pattern of folding. (Examples include effects of chaperone molecules and prions.) (See pages 1003 and 1184.)

Origins of Simple Behavior

■ **Previous approaches.** Before the discoveries in this book, nested and sometimes even repetitive behavior were quite often considered complex, and it was assumed that elaborate theories were necessary to explain them. Most of the theories that have been proposed are ultimately equivalent to what I discuss in this section, though they are usually presented in vastly more complicated ways.

■ **Uniformity in frequency.** As shown on page 587, a completely random sequence of cells yields a spectrum that is essentially uniform in frequency. Such uniformity in frequency is implied by standard quantum theory to exist in the idealized zero-point fluctuations of a free quantum field—with direct consequences for such semiclassical phenomena as the Casimir effect and Hawking radiation. (See page 1062.)

■ **Repetition in numbers.** A common source of repetition in systems involving numbers is the almost trivial fact that in a sequence of successive integers there is a repetitive pattern of cases at which a particular divisor occurs. Other examples include the repetitive structure of digits in rational numbers (see page 138) and continued fraction terms in square roots (see page 144).

■ **Repetition in continuous systems.** A standard approach to partial differential equations (PDEs) used for more than a century is so-called linear stability analysis, in which one assumes that small fluctuations around some kind of basic solution can be treated as a superposition of waves of the form $Exp[i\,k\,x]\,Exp[i\,\omega\,t]$. And at least in a linear approximation any given PDE then typically implies that ω is connected to the wavenumber k by a so-called dispersion relation, which often has a simple algebraic form. For some k this yields a value of ω that is real—corresponding to an ordinary wave that maintains the same amplitude. But for some k one often finds that ω has an imaginary part. The most common case $Im[\omega] > 0$ yields exponential damping. But particularly when the original PDE is nonlinear one often finds that $Im[\omega] < 0$ for some range of k—implying an instability which causes modes with certain spatial wavelengths to grow. The mode with the most negative $Im[\omega]$ will grow fastest, potentially leading to repetitive behavior that shows a particular dominant spatial wavelength. Repetitive patterns with this type of origin are seen in a number of situations, especially in fluids (and notably in connection with Kelvin-Helmholtz, Rayleigh-Taylor and other well-studied instabilities). Examples are ripples and swell on an ocean (compare page 1001), Bénard convection cells, cloud streets and splash coronas. Note that modes that grow exponentially inevitably soon become too large for a linear approximation—and when this approximation breaks down more complicated behavior with no sign of simple repetitive patterns is often seen.

■ **Examples of nesting.** Examples in which a single element splits into others include branching in plants, particle showers, genealogical trees, river deltas and crushing of rocks. Examples in which elements merge include river tributaries and some cracking phenomena.

■ **Page 358 · Nesting in numbers.** Chapter 4 contains several examples of systems based on numbers that exhibit nested behavior. Ultimately these examples can usually be traced to

nesting in the pattern of digits of successive integers, but significant translation is often required.

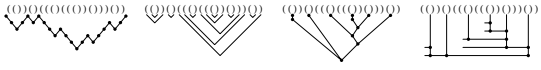
▪ **Nested lists.** One can think of structures that annihilate in pairs as being like parentheses or other delimiters that come in pairs, as in the picture below.

A string of balanced parentheses is analogous to a nested *Mathematica* list such as *{{{}, {{}}}, {}}*. The *Mathematica* expression tree for this list then has a structure analogous to the nested pattern in the picture.

The set of possible strings of balanced parentheses forms a context-free language, as discussed on page 939. The number of such strings containing $2n$ characters is the n^{th} Catalan number *Binomial[2 n, n]/(n + 1)* (as obtained from the generating function *(1 - Sqrt[1 - 4 x])/(2 x)*). The number of strings of depth d (and thus taking d steps to annihilate completely) is given by *c[{n, n}, d] - c[{n, n}, d - 1]* where

```
c[{_, _}, -1] = 0; c[{0, 0}, _] = 1; c[{m_, n_}, _] := 0 /; n > m;
c[{m_, n_}, d_] :=
  Sum[c[{i, j}, d], {i, 0, m - 1}, {j, m - d, n - 1}]
```

Several types of structures are equivalent to strings of balanced parentheses, as illustrated below.

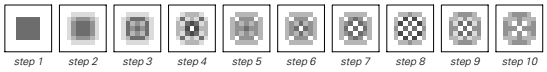


▪ **Phase transitions.** Nesting in systems like rule 184 (see page 273) is closely related to the phenomenon of scaling studied in phase transitions and critical phenomena since the 1960s. As discussed on page 983 ordinary equilibrium statistical mechanics effectively samples configurations of systems like rule 184 after large numbers of steps of evolution. But the point is that when the initial number of black and white cells is exactly equal—corresponding to a phase transition point—a typical configuration of rule 184 will contain domains with a nested distribution of sizes. The properties of such configurations can be studied by considering invariance under rescalings of the kind discussed on page 955, in analogy to renormalization group methods. A typical result is that correlations between colors of different cells fall off like a power of distance—with the specific power depending only on general features of the nested patterns formed, and not on most details of the system.

▪ **Self-organized criticality.** The fact that in traditional statistical mechanics nesting had been encountered only at the precise locations of phase transitions led in the 1980s to the notion that despite its ubiquity in nature nesting must somehow require fine tuning of parameters. Already in the early 1980s, however, my studies on simple additive and other cellular automata (see page 26) had for example made it rather clear that this is not the case. But in the late 1980s it became popular to think that in many systems nesting (as well as the largely unrelated phenomenon of $1/f$ noise) might be the result of fine tuning of parameters achieved through some automatic process of self-regulation. Computer experiments on various cellular automata and related systems were given as examples of how this might work. But in most of these experiments mistakes and misinterpretations were found, and in the end little of value was learned about the origins of nesting (or $1/f$ noise). Nevertheless, a number of interesting systems did emerge, the best known being the idealized sandpile model from the 1987 work of Per Bak, Chao Tang and Kurt Wiesenfeld. This is a $k = 8$ 2D cellular automaton in which toppling of sand above a critical slope is captured by updating an array of relative sand heights s according to the rule

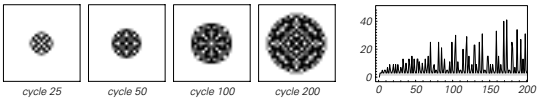
```
SandStep[s_] := s + ListConvolve[
    {{0, 1, 0}, {1, -4, 1}, {0, 1, 0}}, UnitStep[s - 4], 2, 0]
```

Starting from any initial condition, the rule eventually yields a fixed configuration with all values less than 4, as in the picture below. (With an $n \times n$ initial block of 4's, stabilization typically takes about $0.4\, n^2$ steps.).



To model the pouring of sand into a pile one can consider a series of cycles, in which at each cycle one first adds 4 to the value of the center cell, then repeatedly applies the rule until a new fixed configuration *FixedPoint[SandStep, s]* is obtained. (The more usual version of the model adds to a random cell.) The picture below shows slices through the evolution at several successive cycles. Avalanches of different sizes occur, yielding activity that lasts for varying numbers of steps.

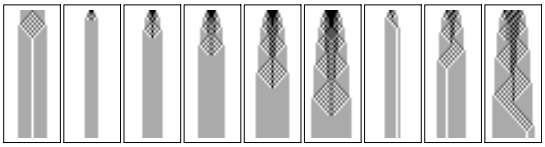
The pictures at the top of the next page show some of the final fixed configurations, together with the number of steps needed to reach them. (The total value of s at cycle t is $4t$; the radius of the nonzero region is about $0.74\sqrt{t}$.) The behavior one sees is fairly complicated—a fact which in the past resulted in much confusion and some bizarre claims, but which in the light of the discoveries in this book no longer seems surprising.



The system can be generalized to d dimensions as a $k = 4\,d$ cellular automaton with $2\,d$ final values. The total value of s is always conserved. In 1D, the update rule is simply

```
SandStep[s_] :=
  s + ListConvolve[{1, -2, 1}, UnitStep[s - 2], 2, 0]
```

In this case the evolution obtained if one repeatedly adds to the center cell (as in the first picture below) is always quite simple. But as the pictures below illustrate, evolution from typical initial conditions yields behavior that often looks a little like rule 184. With a total initial s value of m, the number of steps before a fixed point is reached seems to increase roughly like m^2.



When $d > 1$, more complicated behavior is seen for evolution from at least some initial conditions, as indicated above.

▪ **Random walks.** It is a consequence of the Central Limit Theorem that the pattern of any random walk with steps of bounded length (see page 977) must have a certain nested or self-similar structure, in the sense that rescaled averages of different numbers of steps will always yield patterns that look qualitatively the same. As emphasized by Benoit Mandelbrot in connection with a variety of systems in nature, the same is also true for random walks whose step lengths follow a power-law distribution, but are unbounded. (Compare page 969.)

▪ **Structure of algorithms.** The two most common overall frameworks that have traditionally been used in algorithms in computer science are iteration and recursion—and these correspond quite directly to having operations performed respectively in repetitive and nested ways. But while iteration is generally viewed as being quite easy to understand, until recently even recursion was usually considered rather difficult. No doubt the methods of this book will in the future lead to all sorts of algorithms based on much more complex patterns of behavior. (See page 1142.)

▪ **Origins of localized structures.** Much as with other features of behavior, one can identify several mechanisms that can lead to localized structures. In 1D, localized structures sometimes arise as defects in largely repetitive behavior, or more generally as boundaries between states with different properties—such as the different phases of the repetitive background in rule 110. In higher dimensions a common source—especially in systems that show some level of continuity—are point, line or other topological defects (see page 1045), of which vortices are a typical example.

Implications for Everyday Systems

Issues of Modelling

▪ **Page 363 · Uncertainties of this chapter.** In earlier chapters of this book what I have said can mostly be said with absolute certainty, since it is based on observations about the behavior of purely abstract systems that I have explicitly constructed. But in this chapter, I study actual systems that exist in nature, and as a result, most of what I say cannot be said with any absolute certainty, but instead must involve a significant component of hypothesis. For I no longer control the basic rules of the systems I am studying, and instead I must just try to deduce these rules from observation—with the potential that despite my best efforts my deductions could simply be incorrect.

▪ **Experiences of modelling.** Over the course of the past 25 years I have constructed an immense number of models for a wide range of scientific, technical and business purposes. But while these models have often proved extremely useful in practice, I have usually considered them intellectually quite unsatisfactory. For being models, they are inevitably incomplete, and it is never in any definitive sense possible to establish their validity.

▪ **Page 363 · Notes on this chapter.** Much of this book is concerned with topics that have never been discussed in any concrete form before, so that between the main text and these notes I have been able to include a large fraction of everything that is known about them. But in this chapter (as well as some of the ones that follow) the systems I consider have often had huge amounts written about them before, making any kind of complete summary quite impossible.

▪ **Material for this chapter.** Like the rest of this book, this chapter is strongly based on my personal work and observations. For almost all of the systems discussed I have personally collected extensive data and samples, often over the course of many years, and sometimes in quite unlikely and amusing circumstances. I have also tried to study the existing scientific literature, and indeed in working on this chapter I have looked at many thousands of papers and books—even though the vast majority of them tend to ignore overall issues, and instead concentrate on details of often excruciating specificity.

▪ **Page 365 · Models versus experiments.** In modern science it is usually said that the ultimate test of any model is its agreement with experiment. But this is often interpreted to mean that if an experiment ever disagrees with a model, then the model must be wrong. Particularly when the model is simple and the experiment is complex, however, my personal experience has been that it is quite common for it to be the experiment, rather than the model, that is wrong. When I started doing particle physics in the mid-1970s I assumed—like most theoretical scientists—that the results of experiments could somehow always be treated as rigid constraints on models. But in 1977 I worked on constructing the first model based on QCD for heavy particle production in high-energy proton-proton collisions. The model predicted a certain rate for the production of such particles. But an experiment which failed to see any of these particles implied that the rate must be much lower. And on the basis of this I spent great effort trying to see what might be wrong with the model—only to discover some time later that in fact the methodology of the experiment was flawed and its results were wrong. At first I thought that perhaps this was an isolated incident. But soon I had seen many examples where the stated results of physics experiments were incorrect, either through straightforward mistakes or through subtly prejudiced analysis. And outside of physics, I have tended to find still less reliability in the results of complex experiments.

▪ **Page 366 · Models versus reality.** Questions about the correspondence between models and reality have been much debated in the philosophy of science for many centuries, and were, for example, central to the disagreement between Galileo and the church in the early 1600s. Many successful

models are in practice first introduced as convenient calculational devices, but later turn out to have a direct correspondence to reality. Two examples are planets orbiting the Sun, and quarks being constituents of particles. It remains to be seen whether such models as the imaginary time statistical mechanics formalism for quantum mechanics (see page 1061) turn out to have any direct correspondence to reality.

■ **History of modelling.** Creation myths can in a sense be viewed as primitive models. Early examples of models with more extensive structure included epicycles. Traditional mathematical models of the modern type originated in the 1600s. The success of such models in physics led to attempts to imitate them in other fields, but for the most part these did not succeed. The idea of modelling intricate patterns using programs arose to some extent in the study of fractals in the late 1970s. And the notion of models based on simple programs such as cellular automata was central to my work in the early 1980s. But despite quite a number of fairly well-known successes, there is even now surprisingly little understanding among most scientists of the idea of models based on simple programs. Work in computer graphics—with its emphasis on producing pictures that look right—has made some contributions. And it seems likely that the possibility of computerized and especially image-based data taking will contribute further. (See also page 860.)

■ **Page 367 · Finding models.** Even though a model may have a simple form, it may not be at all easy to find. Indeed, many of the models in this chapter took me a very long time to find. By far my most common mistake was trying to build too much into the basic structure of the model. Often I was sure that some feature of the behavior of a system must be built into the underlying model—yet I could see no simple way to do it. But eventually what happened was that I tried a few other very simple models, and to my great surprise one of them ended up showing the behavior I wanted, even though I had in no way explicitly built it in.

■ **Page 369 · Consequences of models.** Given a program it is always possible to run the program to find out what it will do. But as I discuss in Chapter 12, when the behavior is complex it may take an irreducible amount of computational work to answer any given question about it. However, this is not a sign of imperfection in the model; it is merely a fundamental feature of complex behavior.

■ **Universality in models.** With traditional models based on equations, it is usually assumed that there is a unique correct version of any model. But in the previous chapter we saw that it is possible for quite different programs to yield essentially the same large-scale behavior, implying that with programs there can be many models that have the same consequences but different detailed underlying structure.

The Growth of Crystals

■ **Page 369 · Nucleation.** In the absence of container walls or of other objects that can act as seeds, liquids and gases can typically be supercooled quite far below their freezing points. It appears to be extremely unlikely for spontaneous microscopic fluctuations to initiate crystal growth, and natural snowflakes, for example, presumably nucleate around dust or other particles in the air. Snowflakes in man-made snow are typically nucleated by synthetic materials. In this case and in experiments on cloud seeding it has been observed that the details of seeds can affect the overall shapes of crystals that grow from them.

■ **Page 369 · Implementation.** One can treat hexagonal lattices as distorted square lattices, updated according to

```
CAStep[rule_List, a_] := Map[rule[[14 - #]] &,
    a + 2 ListConvolve[{{1, 1, 0}, {1, 0, 1}, {0, 1, 1}}, a, 2], {2}]
```

where `rule = IntegerDigits[code, 2, 14]`. On this page the rule used is code 16382; on page 371 it is code 10926. The centers of an array of regular hexagons are given by `Table[{i Sqrt[3], j}, {i, 1, m}, {j, Mod[i, 2], n, 2}]`.

■ **Page 372 · Identical snowflakes.** The widespread claim that no two snowflakes are alike is not in practice true. It is however the case that as a result of turbulent air currents a collection of snowflakes that fall to the ground in a particular region will often have come from very different regions of a cloud, and therefore will have grown in different environments. Note that the reason that the six arms of a single snowflake usually look the same is that all of them have grown in essentially the same environment. Deviations are usually the result of collisions between falling snowflakes.

■ **History of snowflake studies.** Rough sketches of snowflakes were published by Olaus Magnus of Uppsala around 1550. Johannes Kepler made more detailed pictures and identified hexagonal symmetry around 1611. Over the course of the next few centuries, following work by René Descartes, Robert Hooke and others, progressively more accurate pictures were made and correlations between weather conditions and snowflake forms were found. Thousands of photographs of snowflakes were taken by Wilson Bentley over the period 1884–1931. Beginning in 1932 an extensive study of snowflakes was made by Ukichiro Nakaya, who in 1936 also produced the first artificial snowflakes. Most of the fairly

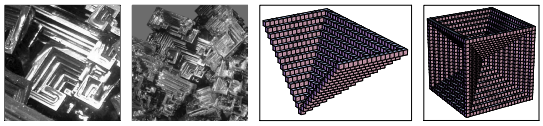
small amount of more recent work on snowflakes has been done as part of more general studies on dendritic crystal growth. Note that tree-like snowflakes are what make snow fluffy, while simple hexagons make it denser and more slippery. The proportion of different types of snowflakes is important in understanding phenomena such as avalanches.

▪ **History of crystal growth.** The vast majority of work done on crystal growth has been concerned with practical methods rather than with theoretical analyses. The first synthetic gemstones were made in the mid-1800s, and methods for making high-quality crystals of various materials have been developed over the course of the past century. Since the mid-1970s such crystals have been crucial to the semiconductor industry. Systematic studies of the symmetries of crystals with flat facets began in the 1700s, and the relationship to internal structure was confirmed by X-ray crystallography in the 1920s. The many different possible external forms of crystals have been noted in mineralogy since Greek times, but although classification schemes have been given, these forms have apparently still not been studied in a particularly systematic way.

▪ **Models of crystal growth.** There are two common types of models for crystal growth: ones based on the physics of individual atoms, and ones based on continuum descriptions of large collections of atoms. In the former category, it was recognized in the 1940s that a single atom is very unlikely to stick to a completely flat surface, so growth will always tend to occur at steps on a crystal surface, often associated with screw dislocations in the crystal structure. In practice, however, as scanning tunnelling microscopes have revealed, most crystal surfaces that are not grown at an extremely slow rate tend to be quite rough at an atomic scale—and so it seems that for example the aggregation model from page 331 may be more appropriate. In snowflakes and other crystals features such as the branches of tree-like structures are much larger than atomic dimensions, so a continuum description can potentially be used. It is possible to write down a nonlinear partial differential equation for the motion of the solidification front, taking into account basic thermodynamic effects. The first result (discovered by William Mullins and Robert Sekerka in 1963) is that if every part of the front is at the same temperature, then any deviations from planarity in the front will tend to grow. The shape of the front is presumably stabilized by the Gibbs-Thomson effect, which implies that the freezing temperature is lower when the front is more curved. The characteristic length for deformations of the front turns out to be the geometric mean of a microscopic length associated with surface energy and a macroscopic length associated with diffusion. It is this characteristic length that presumably determines the size of an individual cell in the cellular automaton model.

Dendritic crystals are commonly seen in ice formations on windows, and in pieces of aluminum of the kind found at typical hardware stores.

▪ **Hopper crystals.** When a pool of molten bismuth solidifies it tends to form crystals like those in the first two pictures below. What seems to give these crystals their characteristic "hoppered" shapes is that there is more rapid growth at the edges of each face than at the center. (Spirals are probably associated with underlying screw dislocations.) Hoppering has not been much studied for scientific purposes, but has been noticed in many substances, including galena, rose quartz, gold, calcite, salt and ice.



▪ **Page 373 · Other models.** There are many ways to extend the simple cellular automata shown here. One possibility is to allow dependence on next-nearest as well as nearest neighbors. In this case it turns out that non-convex as well as convex faceted shapes can be obtained. Another possibility is to allow cells that have become black to turn white again. In this case all the various kinds of patterns that we saw in Chapter 5 can occur. A general feature of cellular automaton rules is that they are fundamentally local. Some models of crystal growth, however, call for long-range effects such as a temperature field which changes throughout the crystal in an effectively instantaneous way. It turns out, however, that many seemingly long-range effects can actually be captured quite easily in cellular automata. In a typical case, this can be done by introducing a third possible color for each cell, and then having rapidly changing arrangements of this color.

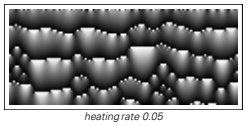
▪ **Polycrystalline materials.** When solids with complicated forms are seen, it has usually been assumed that they must be aggregates of many separate crystals, with each crystal having a simple faceted shape. But the results given here indicate that in fact individual crystals can yield highly complex shapes. There will nevertheless be cases however where multiple crystals are involved. These can be modelled by having a cellular automaton in which one starts from several separated seeds. Sometimes the regions associated with different seeds may have different characteristics; the boundaries between these regions then form a Voronoi diagram (see page 1038).

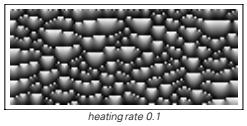
■ **Quasicrystals.** In some special materials it was discovered in 1984 that atoms are arranged not on a purely repetitive grid, but instead in a pattern with the nested type of structure discussed on page 932. A characteristic feature of such patterns is that they can have approximate pentagonal or icosahedral symmetry, which is impossible for purely repetitive patterns. It has usually been assumed that the arrangement of atoms in a quasicrystal is determined by satisfying a constraint analogous to minimization of energy. And as we saw on page 932 it is indeed possible to get nested patterns by requiring that certain constraints be satisfied. But another explanation for such patterns is that they are the result of growth processes that are some kind of cross between those on pages 373 and 659.

■ **Amorphous materials.** When solidification occurs fairly slowly, atoms have time to arrange themselves in a regular crystalline way. But if the cooling is sufficiently rapid, amorphous solids such as glasses are often formed. And in such cases, the packing of atoms is quite random—except that locally there is often approximate icosahedral structure, analogous to that discussed on page 943. (See also page 986.)

■ **Diffusion-limited aggregation (DLA).** DLA is a model for a variety of natural growth processes that was invented by Thomas Witten and Leonard Sander in 1981, and which at first seems quite different from a cellular automaton. The basic idea of DLA is to build up a cluster of black cells by starting with a single black cell and then successively introducing new black cells far away that undergo random walks and stick to the cluster as soon as they come into contact with it. The patterns that are obtained by this procedure turn out for reasons that are still not particularly clear to have a random but on average nested form. (Depending on precise details of the underlying model, very large clusters may sometimes not have nested forms, at least in 2D.) The basic reason that DLA patterns are not very dense is that once arms have formed on the outside of the cluster, they tend to catch new cells before these cells have had a chance to go inside. It turns out that at a mathematical level DLA can be reproduced by solving the Laplace equation at each step with a constant boundary condition on the cluster, and then using the result to give the probability for adding a new cell at each point on the cluster. To construct a cellular automaton analog of DLA one can introduce gray as well as black and white cells, and then have the gray cells represent pieces of solid that have not yet become permanently attached to the main cluster. Rapid rearrangement of gray cells on successive steps can then have a similar effect to the random walks that occur in the usual DLA model. Whether a pattern with all the properties expected in DLA is produced seems to depend in some detail on the rules for the gray cells. But so long as there is effective randomness in the successive positions of these cells, and so long as the total number of them is conserved, then it appears that DLA-like results are usually obtained. No doubt there are also simpler cellular automaton rules that yield similar results. (See also page 979.)

■ **Boiling.** The boiling of a liquid such as water involves a kind of growth inhibition that is in some ways analogous to that seen in dendritic crystal growth. When a particular piece of liquid boils—forming a bubble of gas—a certain latent heat is consumed, reducing the local temperature, and inhibiting further boiling. In the pictures below the liquid is divided into cells, with each cell having a temperature from 0 to 1, corresponding exactly to a continuous cellular automaton of the kind discussed on page 155. At each step, the temperature of every cell is given by the average of its temperature and the temperatures of its neighbors, representing the process of heat diffusion, with a constant amount added to represent external heating. If the temperature of any cell exceeds 1, then only the fractional part is kept, as in the systems on page 158, representing the consumption of latent heat in the boiling process. The pictures below illustrate the kind of seemingly random pattern of bubble formation that can be heard in the noise produced by boiling water.


heating rate 0.05


heating rate 0.1

The Breaking of Materials

■ **Phenomenology of microscopic fracture.** Different materials show rather different characteristics depending on how ductile or brittle they are. Ductile materials—such as taffy or mild steel—bend and smoothly neck before breaking. Brittle materials—such as chalk or glass—do not deform significantly before catastrophic failure. Ductile materials in effect flow slightly before breaking, and as a result their fracture surfaces tend to be less jagged. In addition, in response to stresses in the material, small voids often form—perhaps nucleating around imperfections—yielding a pock-marked surface. In brittle materials, the beginning of the fracture surface typically looks quite mirror-like, then it starts to look misty, and finally, often at a sharply defined point, it begins to look complex and hackled. (This sequence is qualitatively not unlike the initiation of randomness in turbulent fluid flow and many other systems.) Cracks in

brittle materials typically seem to start slowly, then accelerate to about half the Rayleigh speed at which small deformation waves on the surface would propagate. Brittle fracture involves violent breaking of atomic bonds; it usually leaves a jagged surface, and can lead to emission of both high-frequency sound as well as light. Directly around a crack complex patterns of stress are typically produced, though away from the crack they resolve quickly to a fairly smooth and simple form. It is known that ultrasound can affect the course of cracks, suggesting that crack propagation is affected by local stresses. There are many different detailed geometries for fracture, associated with snapping, tearing, shattering, pulling apart, and so on. In many situations, individual cracks will split into multiple cracks as they propagate, sometimes producing elaborate tree-like structures. The statistical properties of fracture surfaces have been studied fairly extensively. There is reasonable evidence of self-similarity, typically associated with a fractal dimension around 0.8 or slightly smaller.

▪ **Models of microscopic fracture.** Two kinds of models have traditionally been studied: ones based on looking at arrays of atoms, and ones based on continuum descriptions of materials. At the atomic level, a simple model suggested fairly recently is that atoms are connected by bonds with a random distribution of strengths, and that cracks follow paths that minimize the total strength of bonds to be broken. It is not clear why in a crystal bonds should be of different strengths, and there is some evidence that this model yields incorrect predictions for the statistical properties of actual cracks. A slightly better model, related to the one in the main text, is that the bonds between atoms are identical, and act like springs which break when they are stretched too far. In recent years, computer simulations with millions of atoms have been carried out—usually with realistic but complicated interatomic force laws—and some randomness has been observed, but its origins have not been isolated. A set of nonlinear partial differential equations known as the Lamé equations are commonly used as a continuum description of elastic materials. Various instabilities have been found in these equations, but the equations are based on small deformations, and presumably cannot be relied upon to provide information about fracture.

▪ **History.** Fracture has been a critical issue throughout the history of engineering. Its scientific study was particularly stimulated by failures of various types of ships and aircraft in the 1940s and 1950s, and many quantitative empirical results were obtained, so that by the 1960s ductile fracture as an engineering issue became fairly well understood. In the 1980s, ideas about fractals suggested new interpretations of fracture surfaces, and in the past few years, various models of fracture based on ideas from statistical physics have been tried. Atomic-level computer experiments on fracture began in earnest in the late 1980s, but only very recently has it been possible to include enough atoms to even begin addressing questions about the structure of cracks.

▪ **Page 375 · Experimental data.** To investigate the model in the main text requires looking not only at the path of a crack, but also at dislocations of atoms near it. To do this dynamically is difficult, but in a perfect crystal final patterns of dislocations that remain at the edge of a region affected by fracture can be seen for example by electron diffraction. And it turns out that these often look remarkably like patterns made by 1D class 3 cellular automata. (Similar patterns may perhaps also be seen in recent detailed simulations of fracture processes in arrays of idealized atoms.)

▪ **Large-scale fractures.** It is remarkable to what extent very large-scale fractures can look like small-scale ones. If the path of a crack were, say, a perfect random walk, then one might imagine that large-scale cracks could simply be combinations of many small-scale segments. But when one looks at geological systems, for example, the smallest relevant scales for the cracks one sees are certainly no smaller than particles of soil. And as a result, one needs a more general mechanism, not just one that just relates to atoms and molecules.

▪ **Alternate models.** It is straightforward to set up 3-color cellular automata with the same basic idea as in the main text, but in which there is no need for a special cell to represent the crack. In addition, instead of modelling the displacement of atoms, one can try to model directly the presence or absence of atoms at particular positions. And then one can start from a repetitive array of cells, with a perturbation to represent the beginning of the crack.

▪ **Electric breakdown.** Somewhat related to fracture is the process of electric breakdown, visible for example in lightning, Lichtenberg figures or plasma-filled glass globes used as executive toys. At least in the case of lightning, there is some evidence that small inhomogeneities in the atmosphere can be important in producing at least some aspects of the apparent randomness that is seen. (With electric potential thought of like a diffusion field, models based on diffusion-limited aggregation are sometimes used.)

▪ **Crushing.** For a rather wide range of cases it appears that in crushed solids such as rocks the probability of a particular fragment having a diameter larger than r is given approximately by $r^{-2.5}$. It seems likely that the origin of this is

that each rock has a certain probability to break into, say, two smaller rocks at each stage in the crushing process, much as in a substitution system.

▪ **Effects of microscopic roughness.** The two most obvious features that are affected by the microscopic roughness of materials are visual appearance and sliding friction. A perfectly flat surface will reflect light like a mirror. Roughness will lead to more diffuse reflection, although the connection between observed properties of rough surfaces and typical parametrizations used in computer graphics is not clear.

The friction force that opposes sliding is usually assumed to be proportional purely to the force with which surfaces are pressed together. Presumably at least the beginning of the explanation for this slightly bizarre fact is that most of the friction force is associated with microscopic peaks in rough surfaces, and that the number of these peaks that come into close contact increases as surfaces are pushed together.

▪ **Crinkling.** A question somewhat related to fracture concerns the generation of definite creases in crumpled or wrinkled objects such as pieces of paper or fabric. It is not too difficult to make various statements about details of the particular arrangements of creases that can occur, but nothing seems to be known about the origin of the overall randomness that is almost universally seen.

Fluid Flow

▪ **Page 376 · Reynolds numbers.** If a system is to act like a continuum fluid, then almost by definition its behavior can involve only a limited number of macroscopic quantities, such as density and velocity. And from this it follows that patterns of flow should not depend separately on absolute speeds and sizes. Instead, the character of a flow should typically be determined by a single Reynolds number, $Re = U L/\nu$, where U is the characteristic speed of the flow (measured say in *cm/sec*), L is a characteristic size (measured say in *cm*), and ν is the kinematic viscosity of the fluid. For water, $\nu = 0.01$, for air $\nu = 0.15$, and for glycerine $\nu = 10000$, all in units of cm^2/sec. In flow past a cylinder it is conventional to take L to be the diameter of the cylinder. But the fact that the form of flow should depend only on Reynolds number means that in the pictures in the main text for example it is not necessary to specify absolute sizes or speeds: one need only know the product $U L$ that appears in the Reynolds number. In practice, moving one's finger slowly through water gives a Reynolds number of about 100 (so that a regular array of dimples corresponding to eddies are visible behind one's finger), walking in air about 10,000, a boat in the millions, and a large airplane in the billions.

The Reynolds number roughly measures the ratio of inertial to viscous effects. When the Reynolds number is small the viscous damping dominates, and the flow is laminar and smooth. When the Reynolds number is large, inertia associated with fluid motions dominates, and the flow is turbulent and complicated.

In different systems, the characteristic length used typically in the definition of Reynolds number is different. In most cases, however, the transition from laminar to turbulent flow occurs at Reynolds numbers around a hundred.

In some situations, however, Reynolds number alone does not appear to be sufficient to determine when a flow will become turbulent. Indeed, modern experiments on streams of dye in water (or rising columns of smoke) typically show a transition to turbulence at a significantly lower Reynolds number than the original experiments on these systems done by Osborne Reynolds in the 1880s. Presumably the reason for this is that the transition point can be lowered by perturbations from the environment, and such perturbations are more common in the modern mechanized world. If perturbations are indeed important, it implies that a traditional fluid description is not adequate. I suspect, however, that even though perturbations may determine the precise point at which turbulence begins, intrinsic randomness generation will dominate once turbulence has been initiated.

▪ **Navier-Stokes equations.** The traditional model of fluids used in physics is based on a set of partial differential equations known as the Navier-Stokes equations. These equations were originally derived in the 1840s on the basis of conservation laws and first-order approximations. But if one assumes sufficient randomness in microscopic molecular processes they can also be derived from molecular dynamics, as done in the early 1900s, as well as from cellular automata of the kind shown on page 378, as I did in 1985 (see below). For very low Reynolds numbers and simple geometries, it is often possible to find explicit formulas for solutions to the Navier-Stokes equations. But even in the regime of flow where regular arrays of eddies are produced, analytical methods have never yielded complete explicit solutions. In this regime, however, numerical approximations are fairly easy to find. Since about the 1960s computers have been powerful enough to allow computations at least nominally to be extended to considerably higher Reynolds numbers. And indeed it has become increasingly common to see numerical results given far into the turbulent regime—leading

sometimes to the assumption that turbulence has somehow been derived from the Navier-Stokes equations. But just what such numerical results actually have to do with detailed solutions to the Navier-Stokes equations is not clear. For in particular it ends up being almost impossible to distinguish whatever genuine instability and apparent randomness may be implied by the Navier-Stokes equations from artifacts that get introduced through the discretization procedure used in solving the equations on a computer. One of the key advantages of my cellular automaton approach to fluids is precisely that it does not require any such approximations.

At a mathematical level analysis of the Navier-Stokes has never established the formal uniqueness and existence of solutions. Indeed, there is even some evidence that singularities might almost inevitably form, which would imply a breakdown of the equations, and perhaps a need to account for underlying molecular processes.

In turbulent flow at higher Reynolds numbers there begin to be eddies with a wide range of sizes. And to capture all these eddies in a computation eventually involves prohibitively large amounts of information. In practice, therefore, semi-empirical models of turbulence tend to be used—often "eddy viscosities"—with no direct relation to the Navier-Stokes equations. In airflow past an airplane there is however typically only a one-inch layer on each surface where such issues are important; the large-scale features of the remainder of the flow, which nevertheless accounts for only about half the drag on the airplane, can usually be studied without reference to turbulence.

The Navier-Stokes equations assume that all speeds are small compared to the speed of sound—and thus that the Mach number giving the ratio of these speeds is much less than one. In essentially all practical situations, Mach numbers close to one occur only at extremely high Reynolds numbers—where turbulence in any case would make it impossible to work out the detailed consequences of the Navier-Stokes equations. Nevertheless, in the case of cellular automaton fluids, I was able in 1985 to work out the rather complicated next order corrections to the Navier-Stokes equations.

Above the speed of sound, fluids form shocks where density or velocity change over very small distances (see below). And by Mach 4 or so, shocks are typically so sharp that changes occur in less than the distance between molecular collisions—making it essential to go beyond the continuum fluid approximation, and account for molecular effects.

▪ **Models of turbulence.** Traditional models typically view turbulence as consisting of some form of cascade of eddies. This notion was already suggested in pictures by Leonardo da Vinci from around 1510, and in Japanese pictures (notably by Katsushika Hokusai) from around 1800 showing ocean waves breaking into precisely nested tongues of water. The theoretical study of turbulence began in earnest in the early 1900s, with emphasis on issues such as energy transfer among eddies and statistical correlations between velocities. Most published work became increasingly mathematical, but particularly following the ideas of Lewis Richardson in the 1920s, the underlying physical notion was that a large eddy, formed say by fluid flowing around an object, would be unstable, and would break up into smaller eddies, which in turn would break up into still smaller eddies, until eventually the eddies would be of such a size as to be readily damped by viscosity. An important step was taken in 1941 by Andrei Kolmogorov who argued that if the eddies in such a cascade were in a statistical equilibrium, then dimensional analysis would effectively imply that the spectrum of velocity fluctuations associated with the eddies must have a $k^{-5/3}$ distribution, with k being wavenumber. This result has turned out to be in respectable agreement with a range of experimental data, but its physical significance has remained somewhat unclear. For there appear to be no explicit entities in fluids that can be directly identified as cascades of eddies. One possibility might be that an eddy could correspond to a local patch of vorticity or rotation in the fluid. And it is a general feature of fluids that interfaces between regions of different velocity are unstable, typically first becoming wavy and then breaking into separate pieces. But physical experiments and simulations in the past few years have suggested that vorticity in turbulent fluids in practice tends to become concentrated on a complicated network of lines that stretch and twist. Perhaps some interpretation can be made involving eddies existing only in a fractal region, or interacting with each other as well as branching. And perhaps new forms of definite localized structures can be identified. But no clear understanding has yet emerged, and indeed most of the analysis that is done—which tends to be largely statistical in nature—is not likely to shed much light on the general question of why there is so much apparent randomness in turbulence.

▪ **Chaos theory and turbulence.** The full Navier-Stokes equations for fluid flow are far from being amenable to traditional mathematical analysis. But some simplified ordinary differential equations which potentially approximate various situations in fluid flow can be more amenable to analysis—and can exhibit the chaos phenomenon. Work in the 1950s by Lev Landau, Andrei Kolmogorov and others focused on equations with periodic

and quasiperiodic behavior. But in 1962 Edward Lorenz discovered more complicated behavior in computer experiments on equations related to fluid flow (see page 971). Analysis of this behavior was closely linked to the chaos phenomenon of sensitive dependence on initial conditions. And by the late 1970s it had become popular to believe that the randomness in fluid turbulence was somehow associated with this phenomenon.

Experiments in very restricted situations showed correspondence with iterated maps in which the chaos phenomenon is seen. But the details of the connection with true turbulence remained unclear. And as I argue in the main text, the chaos phenomenon in the end seems quite unlikely to explain most of the randomness we see in turbulence. The basic problem is that a complex pattern of flow in effect involves a huge amount of information—and to extract this information purely from initial conditions would require for example going to a submolecular level, far below where traditional models of fluids could possibly apply.

Even within the context of the Lorenz equations there are already indications of difficulties with the chaos explanation. The Lorenz equations represent a first-order approximation to certain Navier-Stokes-like equations, in which viscosity is ignored. And when one goes to higher orders progressively more account is taken of viscosity, but the chaos phenomenon becomes progressively weaker. I suspect that in the limit where viscosity is fully included most details of initial conditions will simply be damped out, as physical intuition suggests. Even within the Lorenz equations, however, one can see evidence of intrinsic randomness generation, in which randomness is produced without any need for randomness in initial conditions. And as it turns out I suspect that despite subsequent developments the original ideas of Andrei Kolmogorov about complicated behavior in ordinary differential equations were probably more in line with my notion of intrinsic randomness generation than with the chaos phenomenon.

▪ **Flows past objects.** By far the most experimental data has been collected for flows past cylinders. The few comparisons that have been done indicate that most results are extremely similar for plates and other non-streamlined or "bluff" objects. For spheres at infinitesimal Reynolds numbers a fairly simple exact analytical solution to the Navier-Stokes equations was found by George Stokes in 1851, giving a drag coefficient of $6\pi/R$. For a cylinder, there are difficulties with boundary conditions at infinity, but the drag coefficient was nevertheless calculated by William Oseen in 1915 to be $8\pi/(R(1/2 + \text{Log}[8/R] - \text{EulerGamma}))$. At infinitesimal Reynolds number the flow around a symmetrical object is always symmetrical. As the Reynolds number increases, it becomes progressively more asymmetrical, and at $R \simeq 6$ for a cylinder, closed eddies begin to appear behind the object. The length of the region associated with these eddies is found to grow almost perfectly linearly with Reynolds number. At $R \simeq 30-40$ for a cylinder, oscillations are often seen in the eddies, and at $R \simeq 46-49$, a vortex street forms. Increasingly accurate numerical calculations based on direct approximations to the Navier-Stokes equations have been done in the regime of attached eddies since the 1930s. For a vortex street no analytical solution has ever been found, and indeed it is only recently that the general paths of fluid elements have even been accurately deduced. A simple model due to Theodore von Kármán from 1911 predicts a relative spacing of $\pi/\text{Log}[1+\sqrt{2}]$ between vortices, and bifurcation theory analyses have provided some justification for some such result. Over the range $50 \lesssim R \lesssim 150$ vortices are found to be generated at a cylinder with almost perfect periodicity at a dimensionless frequency (Strouhal number) that increases smoothly from about 0.12 to 0.19. But even though successive vortices are formed at fixed intervals, irregularities can develop as the array of vortices goes downstream, and such irregularities seem to occur at lower Reynolds numbers for flows past plates than cylinders. Some direct calculations of interactions between vortices have been done in the context of the Navier-Stokes equations, but the cellular automaton approach of page 378 seems to provide essentially the first reliable global results. In both calculations and experiments, there is often sensitivity to details of whatever boundary conditions are imposed on the fluid, even if they are far from the object. Results can also be affected by the history of the flow. In general, the early way the flow develops over time typically mirrors quite precisely the long-time behavior seen at successively greater Reynolds numbers. In experiments, the process of vortex generation at a cylinder first becomes irregular somewhere between $R = 140$ and $R = 194$. After this surprisingly few qualitative changes are seen even up to Reynolds numbers as high as 100,000. There is overall periodicity much like in a vortex street, but the detailed motion of the fluid is increasingly random. Typically the scale of the smallest eddies gets smaller in rough correspondence with the $R^{-3/4}$ prediction of Kolmogorov's general arguments about turbulence. In flow past a cylinder, there are various quite sudden changes in the periodicity, apparently associated with 3D phenomena in which the flow is not uniform along the axis of the cylinder. The drag coefficient remains almost constant at a value around 1 until $R \simeq 3 \times 10^5$, at which point it drops precipitously for a while.

This phenomenon is associated with details of flow close to the cylinder. At lower Reynolds numbers, the flow is still laminar when it first comes around the cylinder; but there is a transition to turbulence in this boundary layer after which the fluid can in effect slide more easily around the cylinder. When the speed of the flow passes the speed of sound in the fluid, shocks appear. Usually they form simple geometrical patterns (see below), and have the effect of forcing the turbulent wake behind the cylinder to become narrower.

■ **2D fluids.** The cellular automaton shown in the main text is purely two-dimensional. Experiments done on soap films since the 1980s indicate, however, that at least up to Reynolds numbers of several hundred, the patterns of flow around objects such as cylinders are almost identical to those seen in ordinary 3D fluids. The basic argument for Kolmogorov's $k^{-5/3}$ result for the spectrum of turbulence is independent of dimension, but there are reasons to believe that in 2D eddies will tend to combine, so that after sufficiently long times only a small number of large eddies will be left. There is some evidence for this kind of process in the Earth's atmosphere, as well as in such phenomena as the Red Spot on Jupiter. At a microscopic level, there are some not completely unrelated issues in 2D about whether perturbations in a fluid made up of discrete molecules damp quickly enough to lead to ordinary viscosity. Formally, there is evidence that the Navier-Stokes equations in 2D might have a $\nabla^2 \text{Log}[\nabla^2]$ viscosity term, rather than a ∇^2 one. But this effect, even if it is in fact present in principle, is almost certainly irrelevant on the scales of practical experiments.

■ **Cellular automaton fluids.** A large number of technical issues can be studied in connection with cellular automaton fluids. Many were already discussed in my original 1985 paper. Others have been covered in some of the many papers that have appeared since then. Of particular concern are issues about how rotation and translation invariance emerge at the level of fluid processes even though they are absent in the underlying cellular automaton structure. The very simplest rules turn out to have difficulties in these regards (see page 1024), which is why the model shown in the main text, for example, is on a hexagonal rather than a square grid (compare page 980). The model can be viewed as a block cellular automaton of the type discussed on page 460, but on a 2D hexagonal grid. In general a block cellular automaton works by making replacements for overlapping blocks of cells on alternating steps. In the 1D case of page 460, the blocks that are replaced consist of pairs of adjacent cells with two different alignments. On a 2D square grid, one can use overlapping 2×2 square blocks. But on a 2D hexagonal grid, one must instead alternate on successive steps between hexagons and their dual triangles.

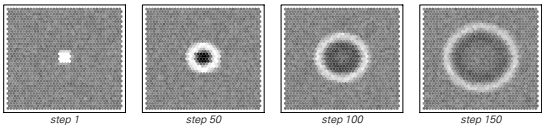
■ **Vorticity-based models.** As an alternative to models of fluids based on elements with discrete velocities, one can consider using elements with discrete vorticities.

■ **History of cellular automaton fluids.** Following the development of the molecular model for gases in the late 1800s (see page 1019), early mathematical derivations of continuum fluid behavior from underlying molecular dynamics were already complete by the 1920s. More streamlined approaches with the same basic assumptions continued to be developed over the next several decades. In the late 1950s Berni Alder and Thomas Wainwright began to do computer simulations of idealized molecular dynamics of 2D hard spheres—mainly to investigate transitions between solids, liquids and gases. In 1967 they observed so-called long-time tails not expected from existing calculations, and although it was realized that these were a consequence of fluid-like behavior not readily accounted for in purely microscopic approximations, it did not seem plausible that large-scale fluid phenomena could be investigated with molecular dynamics. The idea of setting up models with discrete approximations to the velocities of molecules appears to have arisen first in the work of James Broadwell in 1964 on the dynamics of rarefied gases. In the 1960s there was also interest in so-called lattice gases in which—by analogy with spin systems like the Ising model—discrete particles were placed in all possible configurations on a lattice subject to certain local constraints, and average equilibrium properties were computed. By the early 1970s more dynamic models were sometimes being considered, and for example Yves Pomeau and collaborators constructed idealized models of gases in which both positions and velocities of molecules were discrete. As it happens, in 1973, as one of my earliest computer programs, I created a simulation of essentially the same kind of system (see page 17). But it turned out that this particular kind of system, set up as it was on a square grid, was almost uniquely unable to generate the kind of randomness that we have seen so often in this book, and that is needed to obtain standard large-scale fluid behavior. And as a result, essentially no further development on discrete models of fluids was then done until after my work on cellular automata in the early 1980s. I had always viewed turbulent fluids as an important potential application for cellular automata. And in 1984, as part of work I was doing on massively parallel computing, I resolved to develop a practical approach to fluid mechanics based on cellular automata. I initiated discussions with

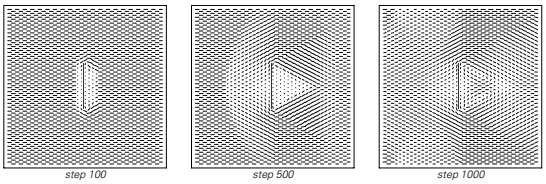
various members of the fluid dynamics community, who strongly discouraged me from pursuing my ideas. But I persisted, and by the summer of 1985 I had managed to produce pictures like those on page 378. Meanwhile, however, some of the very same individuals who had discouraged me had in fact themselves pursued exactly the line of research I had discussed. And by late 1985, cellular automaton fluids were generating considerable interest throughout the fluid mechanics community. Many claims were made that existing computational methods were necessarily far superior. But in practice over the years since 1985, cellular automaton methods have grown steadily in popularity, and are now widely used in physics and engineering. Yet despite all the work that has been done, the fundamental issues about the origins of turbulence that I had originally planned to investigate in cellular automaton fluids have remained largely untouched.

▪ **Computational fluid dynamics.** From its inception in the mid-1940s until the invention of cellular automaton fluids in the 1980s, essentially all computational fluid dynamics involved taking the continuum Navier-Stokes equations and then approximating these equations using some form of discrete mesh in space and time, and arguing that when the mesh becomes small enough, correct results would be obtained. Cellular automaton fluids start from a fundamentally discrete system which can be simulated precisely, and thus avoid the need for any such arguments. One issue however is that in the simplest cellular automaton fluids molecules are in effect counted in unary: each molecule is traced separately, rather than just being included as part of a total number that can be manipulated using standard arithmetic operations. A variety of tricks, however, maintain precision while in effect allowing a large number of molecules to be handled at the same time.

▪ **Sound waves and shocks.** Sound waves in a fluid correspond to periodic variations in density. The pictures below show how a density perturbation leads to a sound wave in a cellular automaton fluid. The sound wave turns out to travel at a fraction $1/\sqrt{2}$ of the microscopic particle speed.



When the speed of a fluid relative to an object becomes comparable to the speed of sound, the fluid will inevitably show variations in density. Typically shocks develop at the front and back of an object, as illustrated below.



It turns out that when two shocks meet, they usually have little effect on each other, and when there are boundaries, shocks are usually reflected in simple ways. The result of this is that in most situations patterns of shocks generated have a fairly simple geometrical structure, with none of the randomness of turbulence.

▪ **Splashes.** Particularly familiar everyday examples of complex fluid behavior are splashes made by objects falling into water. When a water drop hits a water surface, at first a symmetrical crater forms. But soon its rim becomes unstable, and several peaks (often with small drops at the top) appear in a characteristic coronet pattern. If the original drop was moving quickly, a whole hemisphere of water then closes in above. But in any case a peak appears at the center, sometimes with a spherical drop at the top. If a solid object is dropped into water, the overall structure of the splash made can depend in great detail on its shape and surface roughness. Splashes were studied using flash photography by Arthur Worthington around 1900 (as well as Harold Edgerton in the 1950s), but remarkably little theoretical investigation of them has ever been made.

▪ **Generalizations of fluid flow.** In the simplest case the local state of a fluid is characterized by its velocity and perhaps density. But there are many situations where there are also other quantities relevant, notably temperature and chemical composition. And it turns out to be rather straightforward to generalize cellular automaton fluids to handle these.

▪ **Convection.** When there is a temperature difference between the top and bottom of a fluid, hot fluid tends to rise, and cold fluid then comes down again. At low temperature differences (characterized by a low dimensionless Rayleigh number) a regular pattern of hexagonal Bénard convection cells is formed (see page 377). But as the temperature difference increases, a transition to turbulence is seen, with most of the same characteristics as in flow past an object. A cellular automaton model can be made by allowing particles with more than one possible energy: the average particle energy in a region corresponds to fluid temperature.

▪ **Atmospheric turbulence.** Convection occurs because air near the ground is warmer than air at higher altitudes. On a clear night over flat terrain, air flow can be laminar near the ground. Usually, however, it is turbulent near the ground—producing, for example, random gusting in wind—but becomes laminar at higher altitudes. Turbulent convection nevertheless occurs in most clouds, leading to random billowing shapes. The "turbulence" that causes bumps in airplanes is often associated with clouds, though sometimes with larger-scale wave-like fluid motions such as the jet stream.

▪ **Ocean surfaces.** At low wind speeds, regular ripples are seen; at higher wind speeds, a random pattern of creases occurs. It seems likely that randomness in the wind has little to do with the behavior of the ocean surface; instead it is the intrinsic dynamics of the water that is most important.

▪ **Granular materials.** Sand and other granular materials show many phenomena seen in fluids. (Sand dunes are the rough analog of ocean waves.) Vortices have recently been seen, and presumably under appropriate conditions turbulence will eventually also be seen.

▪ **Geological structures.** Typical landscapes on Earth are to a first approximation formed by regions of crust being uplifted through tectonic activity, then being sculpted by progressive erosion (and redeposition of sediment) associated with the flow of water. (Visually very different special cases include volcanos, impact craters and wind-sculpted deserts.) Eventually erosion and deposition will in effect completely smooth out a landscape. But at intermediate times one will see all sorts of potentially dramatic gullies that reflect the pattern of drainage, and the formation of a whole tree of streams and rivers. (Such trees have been studied since at least the early 1900s, with typical examples of concepts being Horton stream order, equal to *Depth* for trees given as *Mathematica* expressions.) If one imagines a uniform slope with discrete streams of water going randomly in each direction at the top, and then merging whenever they meet, one immediately gets a simple tree structure a little like in the pictures at the top of page 359. (More complicated models based for example on aggregation, percolation and energy minimization have been proposed in recent years—and perhaps because most random spanning trees are similar, they tend to give similar results.) As emphasized by Benoit Mandelbrot in the 1970s and 1980s, topography and contour lines (notably coastlines) seem to show apparently random structure on a wide range of scales—with definite power laws being measured in quite a few cases. And presumably at some level this is the result of the nested patterns in which erosion occurs. (An unrelated effect is that as a result of the dynamics of flow in it, even a single river on a featureless landscape will typically tend to increase the curvature of its meanders, until they break off and form oxbow lakes.)

Fundamental Issues in Biology

▪ **Page 383 · History.** The origins of biological complexity have been debated since antiquity. For a long time it was assumed that the magnitude of the complexity was so great that it could never have arisen from any ordinary natural process, and therefore must have been inserted from outside through some kind of divine plan. However, with the publication of Charles Darwin's *Origin of Species* in 1859 it became clear that there were natural processes that could in fact shape features of biological organisms. There was no specific argument for why natural selection should lead to the development of complexity, although Darwin appears to have believed that this would emerge somewhat like a principle in physics. In the century or so after the publication of *Origin of Species* many detailed aspects of natural selection were elucidated, but the increasing use of traditional mathematical methods largely precluded serious analysis of complexity. Continuing controversy about contradictions with religious accounts of creation caused most scientists to be adamant in assuming that every aspect of biological systems must be shaped purely by natural selection. And by the 1980s natural selection had become firmly enshrined as a force of practically unbounded power, assumed—though without specific evidence—to be capable of solving almost any problem and producing almost any degree of complexity.

My own work on cellular automata in the early 1980s showed that great complexity could be generated just from simple programs, without any process like natural selection. But although I and others believed that my results should be relevant to biological systems there was still a pervasive belief that the level of complexity seen in biology must somehow be uniquely associated with natural selection. In the late 1980s the study of artificial life caused several detailed computer simulations of natural selection to be done, and these simulations reproduced various known features of biological evolution. But from looking at such simulations, as well as from my own experiments done from 1980 onwards, I increasingly came to believe that almost any complexity being generated had its origin in phenomena similar to those I had seen in cellular automata—and had essentially nothing to do with natural selection.

▪ **Attitudes of biologists.** Over the years, I have discussed versions of the ideas in this section with many biologists of different kinds. Most are quick to point out at least anecdotal

cases in which features of organisms do not seem to have been shaped by natural selection. But if asked about complexity—either in specific examples or in general—the vast majority soon end up trying to give explanations based on natural selection. Those with a historical bent often recognize that the origins of complexity have always been somewhat mysterious in biology, and indeed sometimes state that this has laid the field open to many attacks. But generally my experience has been that the further one goes from those involved with specific molecular or other details of biological systems the more one encounters a fundamental conviction that natural selection must be the ultimate origin of any important feature of biological systems.

■ **Page 383 · Genetic programs.** Genetic programs are encoded as sequences of four possible nucleotide bases on strands of DNA or RNA. The simplest known viruses have programs that are a few thousand elements in length; bacteria typically have programs that are a few million elements; fruit flies a few hundred million; and humans around four billion. There is not a uniform correspondence between apparent sophistication of organisms and lengths of genetic programs: different species of amphibians, for example, have programs that can differ in length by a factor of a hundred, and can be as many as tens of billions of elements long. Genetic programs are normally broken into sections, many of which are genes that provide templates for making particular proteins. In humans, there are perhaps around 40,000 genes, specifying proteins for about 200 distinct cell types. Many of the low-level details of how proteins are produced is now known, but higher-level issues about organization into different cell types remain somewhat mysterious. Note that although most of the information necessary to construct an organism is encoded in its genetic program, other material in the original egg cell or the environment before birth can probably also sometimes be relevant.

■ **Page 386 · Tricks in evolution.** Among the tricks used are: sexual reproduction, causing large-scale mixing of similar programs; organs, suborganisms, symbiosis and parasitism, allowing different parts of programs to be optimized separately; mutation rate enzymes, allowing parts that need change to be searched more quickly; learnability in individual organisms, allowing larger local deviations from optimality to be tried.

■ **Page 387 · Belief in optimality.** The notion that features of biological organisms are always somehow optimized for a particular purpose has become extremely deep seated—and indeed it has been discussed since antiquity. Most modern biologists at least pay lip service to historical accidents and developmental constraints, but if pressed revert surprisingly quickly to the notion of optimization for a purpose.

■ **Page 390 · Studying natural selection.** From the basic description of natural selection one might have thought that it would correspond to a unique simple program. But in fact there are always many somewhat arbitrary details, particularly centering around exactly how to prune less fit organisms. And the consequence of this is that in my experience it is essentially impossible to come up with precise definitive conclusions about natural selection on the basis of specific simple computer experiments. Using the Principle of Computational Equivalence discussed in Chapter 12, however, I suspect that it will nevertheless be possible to develop a general theory of what natural selection typically can and cannot do.

■ **Page 391 · Other models.** Sequential substitution systems are probably more realistic than cellular automata as models of genetic programs, since elements can explicitly be added to their rules at will. As a rather different approach, one can consider a fixed underlying rule—say a class 4 cellular automaton—with modifications in initial conditions. The notion of universality in Chapter 11 implies that under suitable conditions this should be equivalent to modifications in rules. As an alternative to modelling individual organisms, one can also consider substitution systems which directly generate genealogical trees for populations of organisms, somewhat like Leonardo Fibonacci's original model of a rabbit population.

■ **Page 391 · Adaptive value of complexity.** One might think that the reason complexity is not more widespread in biology is that somehow it is too sensitive to perturbations. But in fact, as discussed in Chapter 7, randomness and complexity tend to lead to more, rather than less, robustness in overall behavior. Indeed, many even seemingly simple biological processes appear to be stabilized by randomness—leading, for example, to random fluctuations in interbeat intervals for healthy hearts. And some biological processes rely directly on complex or random phenomena—for example, finding good paths for foraging for food, avoiding predators or mounting suitable immune responses. (Compare page 1192.)

■ **Page 393 · Genetic algorithms.** As mentioned on page 985, it is straightforward to apply natural selection to computer programs, and for certain kinds of practical tasks with appropriate continuity properties this may be a useful approach.

■ **Page 394 · Smooth variables.** Despite their importance in understanding natural selection both in biology and in potential computational applications, the fundamental

origins of smooth variables or so-called quantitative traits seem to have been investigated rather little. Within populations of organisms such traits are often found to have Gaussian distributions (as, for example, in heights of humans), but this gives little clue as to their origin. (Weights of humans nevertheless have closer to a lognormal distribution.) It is generally assumed that smooth variables must be associated with so-called polygenes that effectively include a large number of individual discrete genes. In pre-Mendelian genetics, observations on smooth variables are presumably what led to the theory that traits of offspring are determined by smoothly mixing the blood of their parents.

▪ **Page 395 · Species.** One feature of biological organisms is that they normally occur in discrete species, with distinct differences between different species. It seems likely that the existence of such discreteness is related to the discreteness of underlying genetic programs. Currently there are a few million species known. Most are distinguished just by their habitats, visual appearance or various simple numerical characteristics. Sometimes, however, it is known that members of different species have the traditional defining characteristic that they cannot normally mate, though this may well be more a matter of the mechanics of mating and development than a fundamental feature.

▪ **Defining life.** See pages 823 and 1178.

▪ **Page 397 · Analogies with thermodynamics.** Over the past century there have been a number of attempts to connect the development of complexity in biological systems with the increase of entropy in thermodynamic systems. In fact, when it was first introduced the very term "entropy" was supposed to suggest an analogy with biological evolution. But despite this, no detailed correspondence between thermodynamics and evolution has ever been forthcoming. However, my statement here that complexity in biology can occur because natural selection cannot control complex behavior is rather similar to my statement in Chapter 9 that entropy can increase because physical experiments in a sense also cannot control complex behavior.

▪ **Page 398 · Major new features.** Traditional groupings of living organisms into kingdoms and phyla are typically defined by the presence of major new features. Standard examples from higher animals include regulation of body temperature and internal gestation of young. Important examples from earlier in the history of life include nuclear membranes, sexual reproduction, multicellularity, protective shells and photosynthesis.

Trilobites are a fairly clear example of organisms where over the course of a few hundred million years the fossil record shows increases in apparent morphological complexity, followed by decreases. Something similar can be seen in the historical evolution of technological systems such as cars.

▪ **Software statistics.** Empirical analysis of the million or so lines of source code that make up *Mathematica* suggests that different functions—which are roughly analogous to different genes—rather accurately follow an exponential distribution of sizes, with a slightly elevated tail.

▪ **Proteins.** At a molecular level much of any living cell is made up of proteins formed from chains of tens to thousands of amino acids. Of the thousands of proteins now known some (like keratin and collagen) are fibrous, and have a simple repetitive underlying structure. But many are globular, and have at least a core in which the 3D packing of amino acids seems quite random. Usually there are some sections that consist of simple α helices, β sheets, or combinations of these. But other parts—often including sites important for function—seem more like random walks. At some level the 3D shapes of proteins (tertiary structure) are presumably determined by energy minimization. But in practice very different shapes can probably have almost identical energies, so that in as much as a given protein always takes on the same shape this must be associated with the dynamics of the process by which the protein folds when it is assembled. (Compare page 988.) One might expect that biological evolution would have had obvious effects on proteins. But as mentioned on page 1184 the actual sequences of amino acids in proteins typically appear quite random. And at some level this is presumably why there seems to be so much randomness in their shapes. (Biological evolution may conceivably have selected for proteins that fold reliably or are more robust with respect to changes in single amino acids, but there is currently no clear evidence for this.)

Growth of Plants and Animals

▪ **History.** The first steps towards a theory of biological form were already taken in Greek times with attempts—notably by Aristotle—to classify biological organisms and to understand their growth. By the 1600s extensive classification had been done, and many structural features had been identified as in common between different organisms. But despite hopes on the part of René Descartes, Galileo and others that biological processes might follow the same kind of rigid clockwork rules that were beginning to emerge in physics, no general principles were forthcoming. Rough analogies between the forms and functions of biological and non-biological systems were fairly common among both artists and scientists, but were rarely thought to

have much scientific significance. In the 1800s more detailed analogies began to emerge, sometimes as offshoots of the field of morphology named by Johann Goethe, and sometimes with mathematical interpretations, and in 1917 D'Arcy Thompson published the first edition of his book *On Growth and Form* which used mathematical methods—mostly from analytical geometry—to discuss a variety of biological processes, usually in analogy with ones in physics. But emphasis on evolutionary rather than mechanistic explanations for a long time caused little further work to be done along these lines. Much additional data was obtained, particularly in embryology, and by the 1930s it seemed fairly clear that at least some aspects of growth in the embryo were controlled by chemical messengers. In 1951 Alan Turing worked out a general mathematical model of this based on reaction-diffusion equations, and suggested that such a model might account for many pigmentation and structural patterns in biological systems (see page 1012). For nearly twenty years, however, no significant follow-up was done on this idea. There were quite a few attempts—often misguided in my opinion—to use traditional ideas from physics and engineering to derive forms of biological organisms from constraints of mechanical or other optimality. And in the late 1960s, René Thom made an important attempt to use sophisticated methods from topology to develop a general theory of biological form. But the mathematics of his work was inaccessible to most natural scientists, and its popularized version, known as catastrophe theory, largely fell into disrepute.

The idea of comparing systems in biology and engineering dates back to antiquity, but for a long time it was mainly thought of just as an inspiration for engineering. In the mid-1940s, however, mostly under the banner of cybernetics, tools from the analysis of electrical systems began to be used for studying biological systems. And partly from this—with much reinforcement from the discovery of the genetic code—there emerged the idea of thinking about biological systems in purely abstract logical or computational terms. This led to an early introduction of 2D cellular automata (see page 876), but the emphasis was on ambitious general questions rather than specific models. Little progress was made, and by the 1960s most work along these lines had petered out. In the late 1970s, however, fractals and L systems (see below) began to provide examples where simple rules could be seen to yield biological-like branching behavior. And in the 1980s, interest in non-equilibrium physical processes, and in phenomena such as diffusion-limited aggregation, led to renewed interest in reaction-diffusion equations, and to somewhat more explicit models for various biological processes. My own work on cellular automata in the early 1980s started a number of new lines of computational modelling, some of which became involved in the rise and fall of the artificial life movement in the late 1980s and early 1990s.

■ **Page 400 · Growth in plants.** At the lowest level, the growth of any organism proceeds by either division or expansion of cells, together with occasional formation of cavities between cells. In plants, cells typically expand—normally through intake of water—only for a limited period, after which the cellulose in their walls crystallizes to make them quite rigid. In most plants—at least after the embryonic stage—cells typically divide only in localized regions known as meristems, and each division yields one cell that can divide again, and one that cannot. Often the very tip of a stem consists of a single cell in the shape of an inverted tetrahedron, and in lower plants such as mosses this is essentially the only cell that divides. In flowering plants, cell division normally occurs around the edge of a region of size 0.2–1 mm containing many tens of cells. (Hearts of palm in palm trees can however be much larger.) The details of how cell division works in plants remain largely unknown. There is some evidence that orientation of new cells is in part controlled by microscopic fibers. Various small molecules that can diffuse between cells (such as so-called auxins) are known to affect growth and production of new stems (see below).

■ **Page 401 · Branching in plants.** Almost all kinds of plants exhibit some form of branching, and particularly in smaller plants the branching is often extremely regular. In a plant as large as a typical tree—particularly one that grows slowly—different conditions associated with the growth of different branches may however destroy some of the regularity of branching. Among algae and more primitive plants such as whisk ferns, repeated splitting of a single branch into two is particularly common. Ferns and conifers both typically exhibit three-way branching. Among flowering plants so-called dicotyledons exhibit branching throughout the plant. Monocotyledons—of which palms and grasses are two examples—typically have only one primary site of growth, and thus do not exhibit repeated branching. (In grasses the growth site is at the bottom of the stem, and in bamboos there are multiple growth sites up the stem.)

The forms of branching in plants have been used as means of classification since antiquity. Alexander von Humboldt in 1808 identified 19 overall types of branching which have been used, with some modifications, by plant geographers and botanists ever since. Note that in the vast majority of

cases, branches do not lie in a plane; often they are instead arranged in a spiral, as discussed on page 408. But when projected into two dimensions, the patterns obtained still look similar to those in the main text.

▪ **Page 402 · Implementation.** It is convenient to represent the positions of all tips by complex numbers. One can take the original stem to extend from the point -1 to 0; the rule is then specified by the list *b* of complex numbers corresponding to the positions of the new tip obtained after one step. And after *n* steps the positions of all tips generated are given simply by

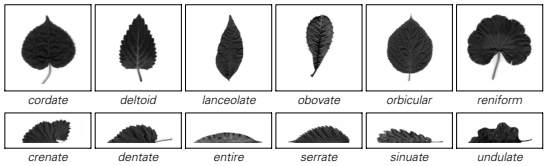
```
Nest[Flatten[Outer[Times, 1 + #, b]] &, {0}, n]
```

▪ **Mathematical properties.** If an element *c* of the list *b* is real, so that there is a stem that goes straight up, then the limiting height of the center of the pattern is obtained by summing a geometric series, and is given by *1/(1 - c)*. The overall limiting pattern will be finite so long as *Abs[c] < 1* for all elements of *b*. After *n* steps the total length of all stems is given by $Apply[Plus, Abs[b]]^n$. (See page 1006 for other properties.)

▪ **Page 402 · Simple geometries.** Page 357 shows how some of the nested patterns commonly seen in this book can be produced by the growth processes shown here.

▪ **History of branching models.** The concept of systematic rules for the way that stems—particularly those carrying flowers—are connected in a plant seems to have been clearly understood among botanists by the 1800s. Only with the advent of computer graphics in the 1970s, however, does the idea appear to have arisen of varying angles to get different forms. An early example was the work of Hisao Honda in 1970 on the structure of trees. Pictures analogous to the bottom row on page 402 were also generated by Benoit Mandelbrot in connection with his development of fractals. Starting in 1967 Aristid Lindenmayer emphasized the use of substitution or L systems (see page 893) as a way of modelling patterns of connections in plants. And beginning in the early 1980s—particularly through work by Alvy Ray Smith and later Przemyslaw Prusinkiewicz—models based on L systems and fractals became routinely used for producing images of plants in practical computer graphics. Around the same time Michael Barnsley also used so-called iterated function systems to make pictures of ferns—but he appears to have viewed these more as a curiosity than a contribution to botany. Over the past decade or so, a few mentions have been made of using complicated models based on L systems to reproduce shapes of specific types of leaves, but so far as I can tell, nothing like the simple model that I describe in the main text has ever been considered before.

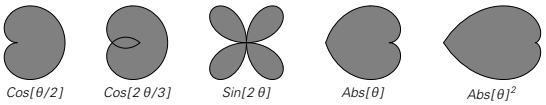
▪ **Page 404 · Leaf shapes.** Leaves are usually put into categories like the ones below, with names mostly derived from Latin words for similar-looking objects.



Some classification of leaf shapes was done by Theophrastus as early as 300 BC, and classifications similar to those above were in use by the early Renaissance period. (They appear for example in the first edition of the *Encyclopedia Britannica* from 1768.) Leaf shapes have been widely used since antiquity as a way of identifying plants—initially particularly for medicinal purposes. But there has been very little general scientific investigation of leaf shapes, and most of what has been done has concentrated on the expansion of leaves once they are out of their buds. Already in 1724 Stephen Hales looked at the motion of grids of marks on fig leaves, and noted that growth seemed to occur more or less uniformly throughout the leaf. Similar but increasingly quantitative studies have been made ever since, and have reported a variety of non-uniformities in growth. For a long time it was believed that after leaves came out of their buds growth was due mainly to cell expansion, but in the 1980s it became clear that many cell divisions in fact occur, both on the boundary and the interior. At the earliest stages, buds that will turn into leaves start as bumps on a plant stem, with a structure that is essentially impossible to discern. Surgically modifying such buds when they are as small as 0.1 mm can have dramatic effects on final leaf shape, suggesting that at least some aspects of the shape are already determined at that point. On a single plant different leaves can have somewhat different shapes—sometimes for example those lower on a tree are smoother, while those higher are pointier. It may nevertheless be that leaves on a single plant initially have a discrete set of possible shapes, with variations in final shape arising from differences in environmental conditions during expansion. My model for leaf shapes is presumably most relevant for initial shapes.

Traditional evolutionary explanations have not had much to say about detailed questions of leaf shape; one minor claim is that the pointed tips at the ends of many tropical leaves exist to allow moisture to drip off the leaves. The fossil record suggests that leaves first arose roughly 400 million years ago, probably when collections of branches which lay in a plane became joined by webbing. Early plants such as ferns have compound leaves in which explicit branching structure is still

seen. Extremely few models for shapes of individual leaves appear to have ever been proposed. In 1917 D'Arcy Thompson mentioned that leaves might have growth rates that are simple functions of angle, and drew the first of the pictures shown below.



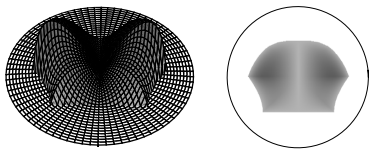
With new tip positions as on page 400 given by *{p Exp[i θ], p Exp[-i θ], q}*, rough *{p, q, θ}* for at least some versions of some common plants include: wild carrot (Queen Anne's lace) *{0.4, 0.7, 30°}*, cypress *{0.4, 0.7, 45°}*, coralbells *{0.5, 0.4, 0°}*, ivy *{0.5, 0.6, 0°}*, grape *{0.5, 0.6, 15°}*, sycamore *{0.5, 0.6, 15°}*, mallow *{0.5, 0.6, 30°}*, goosefoot *{0.55, 0.8, 30°}*, willow *{0.55, 0.8, 80°}*, morning glory *{0.7, 0.8, 0°}*, cucumber *{0.7, 0.8, 15°}*, ginger *{0.65, 0.6, 15°}*.

■ **Page 404 · Self-limiting growth.** It is often said that in plants, unlike animals, there is no global control of growth. And one feature of the simple branching processes I describe is that for purely mathematical reasons, their rules always produce structures that are of limited size. Note that in fact it is known that there is some global control of growth even in plants: for example hormones produced by leaves can affect growth of roots.

■ **Page 407 · Parameter space sets.** Points in the space of parameters can conveniently be labelled by a complex number *c*, where the imaginary direction is taken to increase to the right. The pattern corresponding to each point is the limit of *Nest[Flatten[1 + {c #, Conjugate[c] #}] &, {1}, n]* when $n \to \infty$. Such a limiting pattern exists only within the unit circle *Abs[c] < 1*. It then turns out that the limiting pattern is either completely connected or completely disconnected; which it is depends on whether it contains any points on the vertical axis *Im[c] == 0*. Every point in the pattern must correspond to some list of left and right branchings, represented by 0's and 1's respectively; in terms of this list the position of the point is given by *Fold[1 + {c, Conjugate[c]}[[1 + #2]] #1 &, 1, Reverse[list]]*. Patterns are disconnected if there is a gap between the parts obtained from lists starting with 0 and with 1. The magnitude of this gap turns out to be given by

```
With[{d = Conjugate[c], r = 1 - Abs[c]^2},
  Which[Im[c] < 0, d, Im[c] == 0, 0,
    Re[c] > 0, With[{n = Ceiling[π/2/Arg[c]]},
      Im[c (1 - d^n)/(1 - d)] + Im[c d^n (1 + d)]/r], Arg[c] >
      3 π/4, Im[c + c^2]/r, True, Im[c] + Im[c^2 + c^3]/r]]
```

The picture below shows the region for which the gap is positive, corresponding to trees which are not connected. (This region was found by Michael Barnsley and others in the late 1980s.) The overall maximum gap occurs at *c = 1/2 Sqrt[5 - √17] i*. The bottom boundary of the region lies along *Re[c] = -1/2*; the extremal point on the edge of the gap in this case corresponds to *{0, 0, 1, 0, 1, 0, 1, …}* where the last two elements repeat forever. The rest of the boundary consists of a sequence of algebraic curves, with almost imperceptible changes in slope in between; the first corresponds to *{0, 0, 0, 1, 0, 1, 0, 1, …}*, while subsequent ones correspond to *{0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, …}*, *{0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, …}*, etc.



In the pictures in the main text, the black region is connected wherever it does not protrude into the shaded region, which corresponds to disconnected patterns, in the pictures above. And in general it turns out that near any particular value of *c* the sets shown in black in the main text always look at sufficient magnification like the pattern that would be obtained for that value of *c*. The reason for this is that if *c* changes only slightly, then the pattern to a first approximation deforms only slightly, so that the part seen through the peephole just shifts, and in a small region of *c* values the peephole in effect simply scans over different parts of the pattern.

A simple way to approximate the pictures in the main text would be to generate patterns by iterating the substitution system a fixed number of times. In practice, however, it is essential to prune the tree of points at each stage. And at least for *Abs[c]* not too close to 1, this can be done by discarding points that are so far away from the peephole that their descendents could not possibly return to it.

The parameter space sets discussed here are somewhat analogous to the Mandelbrot set discussed on page 934, though in many ways easier to understand.

(See also the discussion of universal objects on page 1127.)

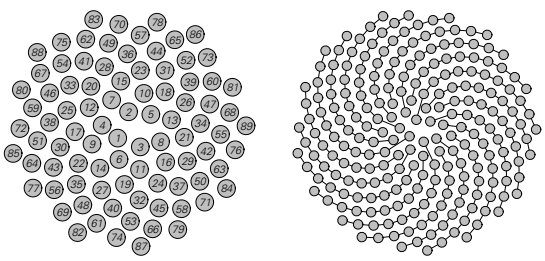
■ **Page 409 · Mathematics of phyllotaxis.** A rotation by *GoldenRatio == (1 + √5)/2* turns is equivalent to a rotation by *2 - GoldenRatio == GoldenRatio^-2* ≃ *0.38* turns, or 137.5°. Successive approximations to this number are given by *Fibonacci[n - 2]/Fibonacci[n]*, so that elements numbered *Fibonacci[n]* (i.e. 1, 2, 3, 5, 8, 13, …) will be the ones that come closest to being a whole number of turns apart, and thus to being lined up on the stem. As mentioned on page 891, having *GoldenRatio* turns between elements makes them in a

sense as evenly distributed as possible, given that they are added sequentially.

■ **History of phyllotaxis.** The regularities of phyllotaxis were presumably noticed in antiquity, and were certainly recognized in the 1400s, notably by Leonardo da Vinci. By the 1800s various mathematical features of phyllotaxis were known, and in 1837 Louis and Auguste Bravais identified the presence of a golden ratio angle. In 1868 Wilhelm Hofmeister proposed that new elements form in the largest gap left by previous elements. And in 1913 Johannes Schoute argued that diffusion of a chemical creates fields of inhibition around new elements—a model in outline equivalent to mine. In the past century features of phyllotaxis have been rediscovered surprisingly many times, with work being done quite independently both in abstract mathematical settings, and in the context of specific models (most of which are ultimately very similar). One development in the 1990s is the generation of phyllotaxis-like patterns in superconductors, ferrofluids and other physical systems.

■ **Observed phyllotaxis.** Many spiral patterns in actual plants converge to within a degree or less of 137.5 °, though just as in the model in the main text, there are usually deviations for the first few elements produced. The angles are particularly accurate in, for example, flower heads—where it is likely the positions of elements are adjusted by mechanical forces after they are originally generated. Other examples of phyllotaxis-like patterns in biology include the scales of pangolins and surfaces of tooth-like structures in certain kinds of rays and sharks.

■ **Projections of patterns.** The literature of phyllotaxis is full of baroque descriptions of the features of projections of patterns with golden ratio angles. In the pictures below, the n^{th} point has position *(√n {Sin[#], Cos[#]} &)[2 π n GoldenRatio]*, and in such pictures regular spirals or parastichies emanating from the center are seen whenever points whose numbers differ by *Fibonacci[m]* are joined. Note that the tips of many growing stems seem to be approximately paraboloidal, making the n^{th} point a distance $\sqrt{n}$ from the center.



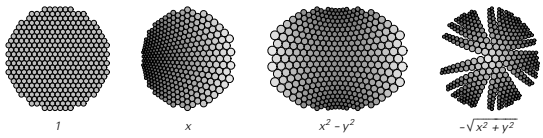
■ **Page 410 · Implementation.** It is convenient to consider a line of discrete cells, much as in a continuous cellular automaton. With a concentrations list *c*, the position *p* of a new element is given by *Position[c, Max[c], 1, 1][[1, 1]]*, while the new list of concentrations is *λ c + RotateRight[f, p]* where *f* is a list of depletions associated with addition of a new element at position 1. In the main text a Gaussian form is used for *f*. Other smooth functions typically nevertheless yield identical results. Note that in order to get an accurate approximation to a golden ratio angle there must be a fairly large number of cells.

■ **Shapes of cells.** Many types of cells are arranged like typical 3D packings of deformable objects (see page 988)—with considerable apparent randomness in individual shapes and positions, but definite overall statistical properties. Cells arranged on a surface—as in the retina or in skin—or that are intrinsically elongated—as in muscle—tend again to be arranged like typical packings, but now in 2D, where a regular hexagonal grid is formed.

■ **Page 412 · Symmetries.** Biological systems often show definite discrete symmetry. (In monocotyledon plants there is usually 3-fold symmetry; in dicotyledons 4- or 5-fold. Animals like starfish often have 5-fold symmetry; higher animals usually only 2-fold symmetry. There are fossils with 7- and 9-fold symmetry. At microscopic levels there are sometimes other symmetries: cilia of eukaryotic cells can for example show 9- and 13-fold symmetry. In the phyllotaxis process discussed in the main text one new element is produced at a time. But if several elements are produced together the same basic mechanism will tend to make these elements be equally spaced in angle—leading to overall discrete symmetry. (Individual proteins sometimes also arrange themselves into overall structures that have discrete symmetries—which can then be reflected in shapes of cells or larger objects.) (See also page 1011.)

■ **Page 412 · Locally isotropic growth.** A convenient way to see what happens if elements of a surface grow isotropically is to divide the surface into a collection of very small circles, and then to expand the circle at each point by a factor *h[x, y]*. If the local curvature of the surface is originally *c[x, y]*, then after such growth, the curvature turns out to be *(c[x, y] + Laplacian[Log[h[x, y]]])/h[x, y]* where *Laplacian[f_] := ∂xx f + ∂yy f*. In order for the surface to stay flat its growth rate *Log[h[x, y]]* must therefore solve Laplace's equation, and hence must be a harmonic function *Re[f[x + i y]]*. This is equivalent to saying that the growth must correspond to a conformal mapping which locally preserves angles. The pictures below show results for several growth rate functions; in the last case, the function

is not harmonic, and the surface cannot be drawn in the plane without tearing. Note that if the elements of a surface are allowed to change shape, then the surface can always remain flat, as in the top row of pictures on page 412. Harmonic growth rate functions can potentially be obtained from the large-time effects of a chemical subject to diffusion. And this may perhaps be related to the flatness observed in the growth of leaves. (See also page 1010.)



■ **Page 413 · Branching in animals.** Capillaries, bronchioles and kidney ducts in higher animals typically seem to form trees in which each tip as it extends repeatedly splits into two branches. (In human lungs, for example, there are about 20 levels of branching.) The same kind of structure is seen in the digestive systems of lower animals—as visible externally, for example, in the arms of a basket star.

■ **Page 413 · Antlers.** Like stems of plants antlers grow at their tips, and can thus exhibit branching. This is made possible by the fact that antlers, unlike horns, have a layer of soft tissue on the outside—which delivers the nutrients needed for growth to occur on the outer surface of the bone at their tips.

■ **Page 414 · Shells.** Shells grow through the secretion of rigid material from the soft lip or mantle of the animal inside, and over periods of months to years they form coiled structures that normally follow rather accurate equiangular spirals, typically right-handed. The number of turns or whorls varies widely, from less than one in a typical bivalve, to more than thirty in a highly pointed univalve such as a screw shell. Usually the coiled structure is obvious from looking at the apex on the outside of the shell, but in cowries, for example, it is made less obvious by the fact that later whorls completely cover earlier ones, and at the opening of the shell some dissolving and resculpting of material occurs. In addition to smooth coiled overall structures, some shells exhibit spines. These are associated with tentacles of tissue which secrete shell material at their tips as they grow. Inside shells such as nautiluses, there are a sequence of sealed chambers, with septa between them laid down perhaps once a month. These septa in present-day species are smooth, but in fossil ammonites they can be highly corrugated. Typically the corrugations are accurately symmetrical, and I suspect that they in effect represent slices through a lettuce-leaf-like structure formed from a surface with tree-like internal growth.

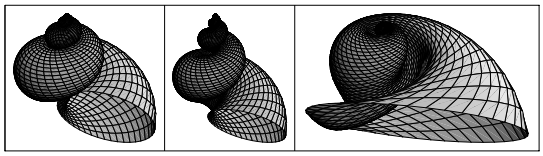
■ **Shell model.** The center of the opening of a shell is taken to trace out a helix whose $\{x, y, z\}$ coordinates are given as a function of the total angle of revolution t by $a^t \{\text{Cos}[t], \text{Sin}[t], b\}$. On row (a) of page 415 the parameter a varies from 1.05 to 1.65, while on row (b) b varies from 0 to 6. The complete surface of the shell is obtained by varying both t and θ in

$$a^t \{\text{Cos}[t] (1 + c (\text{Cos}[e] \text{Cos}[\theta] + d \text{Sin}[e] \text{Sin}[\theta])),\\ \text{Sin}[t] (1 + c (\text{Cos}[e] \text{Cos}[\theta] + d \text{Sin}[e] \text{Sin}[\theta])),\\ b + c (\text{Cos}[\theta] \text{Sin}[e] - d \text{Cos}[e] \text{Sin}[\theta])\}$$

where c varies from 0.4 to 1.6 on row (c), d from 1 to 4 on row (d) and e from 0 to 1.2 on row (e). For many values of parameters the surface defined by this formula intersects itself. However, in an actual shell material can only be added on the outside of what already exists, and this can be represented by restricting θ to run over only part of the range $-\pi$ to π. The effect of this on internal structure can be seen in the slice of the cone shell on row (b) of page 414. Most real shells follow the model described here with remarkable accuracy. There are, however, deviations in some species, most often as a result of gradual changes in parameters during the life of the organism. As the pictures in the main text show, shells of actual molluscs (both current and fossil) exist throughout a large region of parameter space. And in fact it appears that the only parameter values that are not covered are ones where the shell could not easily have been secreted by an animal because its shape is degenerate and leaves little useful room for the animal. Some regions of parameter space are more common than others, and this may be a consequence either of natural selection or of the detailed molecular biology of mollusc growth. Shells where successive whorls do not touch (as in the first picture on row (c) of page 415) appear to be significantly less common than others, perhaps because they have lower mechanical rigidity. They do however occur, though sometimes as internal rather than external shells.

■ **History.** Following Aristotle's notion of gnomon figures that keep the same shape when they grow, equiangular spirals were discussed by René Descartes in 1638, and soon thereafter Christopher Wren noted their relation to shells. A clear mathematical model of shell growth based on equiangular spirals was given by Henry Moseley in 1838, and the model used here is a direct extension of his. Careful studies from the mid-1800s to mid-1900s validated Moseley's basic model for a wide variety of shells, though an increasing emphasis was placed on shells that showed deviations from the model. In the mid-1960s David Raup used early computer graphics to generate pictures for various ranges of parameters, but perhaps because he considered only specific classes of molluscs there emerged from his work the belief

that parameters of shells are greatly constrained—with explanations being proposed based on optimization of such features as strength, relative volume, and stability when falling through water. But as discussed in the main text I strongly suspect that in fact there are no such global constraints, and instead almost all reasonable values of parameters from the simple model used do actually occur in real molluscs. In the past few decades, increasingly complex models for shells have been constructed, typically focusing on fairly specific or unusual cases. Most of these models have far more parameters than the simple one used here, and by varying these parameters it is almost always possible to get forms that probably do not correspond to real shells. And presumably the reason for this is just that such models represent processes that do not occur in the growth of actual molluscs. One widespread issue concerns the orientation of the opening to a shell. The model used here assumes that this opening always stays vertical—which appears to be what happens most often in practice. But following the notion of Frenet frames in differential geometry, it has often come to be supposed that the opening to a shell instead typically lies in a plane perpendicular to the helix traced out by the growth of the shell. This idea, however, leads to twisted shapes like those shown below that occur rarely, if ever, in actual shells. And in fact, despite elaborate efforts of computer graphics it has proved rather difficult with parametrizations based on Frenet frames to produce shells that have a reasonable range of realistic shapes.



▪ **Page 417 · Discrete folding.** See page 892.

▪ **Page 418 · Intrinsically defined curves.** With curvature given by a function $f[s]$ of the arc length s, explicit coordinates $\{x[s], y[s]\}$ of points are obtained from (compare page 1048)

```
NDSolve[{x'[s] == Cos[θ[s]], y'[s] == Sin[θ[s]], θ'[s] ==
    f[s], x[0] == y[0] == θ[0] == 0}, {x, y, θ}, {s, 0, s_max}]
```

For various choices of $f[s]$, formulas for $\{x[s], y[s]\}$ can be found using *DSolve*:

$f[s] = 1$: $\{\text{Sin}[\theta], \text{Cos}[\theta]\}$

$f[s] = s$: $\{\text{FresnelS}[\theta], \text{FresnelC}[\theta]\}$

$f[s] = 1/\sqrt{s}$: $\sqrt{\theta}\ \{\text{Sin}[\sqrt{\theta}\ ], \text{Cos}[\sqrt{\theta}\ ]\}$

$f[s] = 1/s$: $\theta\{\text{Cos}[\text{Log}[\theta]], \text{Sin}[\text{Log}[\theta]]\}$

$f[s] = 1/s^2$: $\theta\{\text{Sin}[1/\theta], \text{Cos}[1/\theta]\}$

$f[s] = s^n$: result involves $\text{Gamma}[1/n, \pm i\,\theta^n/n]$

$f[s] = \text{Sin}[s]$: result involves $\text{Integrate}[\text{Sin}[\text{Sin}[\theta]], \theta]$, expressible in terms of generalized Kampé de Fériet hypergeometric functions of two variables.

When $s_{max} \to \infty$, $f[s] = a\,s\,\text{Sin}[s]$ yields 2D shapes that are basically nested, with pieces overlapping for $\text{Abs}[a] < 1$.

The general idea of so-called natural equations for obtaining curves from local curvature appears to have been first considered by Leonhard Euler in 1736. Many examples with fairly simple behavior were studied in the 1800s. The case of $f[s] = a\,\text{Sin}[b\,s]$ was studied by Eduard Lehr in 1932. Cases related to $f[s] = s\,\text{Sin}[s]$ were studied by Alfred Gray around 1992 using *Mathematica*.

▪ **Multidimensional generalizations.** Curvatures for surfaces and higher-dimensional objects can be defined in terms of the principal axes of approximating ellipsoids at each point. There are combinations of these curvatures—in 2D Gaussian curvature and mean curvature—which are independent of the coordinate system used. (Compare page 1049.) Given such curvatures, a surface can in principle be obtained by solving certain partial differential equations. But even in the case of zero mean curvature, which corresponds to minimal surfaces of the kind followed by an idealized soap film, this is already a mathematical problem of significant difficulty.

If one looks at projections of surfaces, it is common to see lines of discontinuity at which a surface goes, say, from having three sheets to one. Catastrophe theory provides a classification of such discontinuities—the simplest being a cusp. And as emphasized by René Thom in the 1960s, it is possible that some structures seen in animals may be related to such discontinuities.

▪ **Page 419 · Embryo development.** Starting from a single egg cell, embryos first exhibit a series of geometrically quite precise cell divisions corresponding, I suspect, to a simple neighbor-independent substitution system. When the embryo consists of a definite number of cells—from tens to tens of thousands depending on species—the phenomenon of gastrulation occurs, and the hollow sphere of cells that has been produced folds in on itself so as to begin to form more tubular structures. In organisms with a total of just a few thousand cells, the final position and type of every cell seems to be determined directly by the genetic program of the organism; most likely what happens is that each cell division leads to some modification in genetic material, perhaps through rules like those in a multiway system. Beyond a few thousand cells, however, individual cells seem to be less relevant, and instead what appears to happen is that chemicals such as retinoic acid (a derivative of vitamin A)

produced by particular cells diffuse to affect all cells in a region a tenth of a millimeter or so across. Probably as a result of chemical concentration gradients, different so-called homeobox genes are then activated in different parts of the region. Each of these genes—out of a total of 38 in humans—yields proteins which then in turn switch on or off large banks of genes, allowing different forms of behavior for cells in different places.

■ **History of embryology.** General issues of embryology were already discussed in Greek times, notably by Hippocrates and Aristotle. But even in the 1700s it was still thought that perhaps every embryo started from a very small version of a complete organism. In the 1800s, however, detailed studies revealed the progressive development of complexity in the growth of an embryo. At the end of the 1800s experiments based on removing or modifying parts of early embryos began, and by the 1920s it had been discovered that there were definite pieces of embryos that were responsible for inducing various aspects of development to occur. That concentrations of diffusing chemicals might define where in an embryo different elements would form was first suggested in the early 1900s, but it was not until the 1970s and 1980s—after it was emphasized by Lewis Wolpert in 1969 under the name "positional information"—that there was clear experimental investigation of this idea. From the 1930s and before, it was known that different genes are involved in different aspects of embryo development. And with the advent of gene manipulation methods in the 1970s and 1980s, it became possible to investigate the genetic control of development in organisms such as fruit flies in tremendous detail. Among the important discoveries made were the homeobox genes (see note above).

■ **Page 420 · Bones.** Precursors of bones can be identified quite early in the growth of most vertebrate embryos. Typically the cells involved are cartilage, with bone subsequently forming around them. In hardened bones growth normally occurs by replication of cartilage cells in plates perhaps a millimeter thick, with bone forming by a somewhat complicated process involving continual dissolving and redeposition of already hardened material. The rate at which bone grows depends on the pressure exerted on it, and presumably this allows feedback that for example prevents coiling. Quite how the complicated collection of tens of bones that make up a typical skull manage to grow so as to stay connected—often with highly corrugated suture lines—remains fairly mysterious.

■ **General constraints on growth.** Given a system made from material with certain overall properties, one can ask what distributions of growth are consistent with those properties, and what kinds of shapes can be produced. With material that is completely rigid growth can occur only at boundaries. With material where every part can deform arbitrarily any kind of growth can occur. With material where parts can locally expand, but cannot change their shape, page 1007 showed that a 2D surface will remain flat if the growth rate is a harmonic function. The Riemann mapping theorem of complex analysis then implies that even in this case, any smooth initial shape can grow into any other such shape with a suitable growth rate function. In a 3D system with locally isotropic growth the condition to avoid tearing is that the Ricci scalar curvature must vanish, and this is achieved if the local growth rate satisfies a certain partial differential equation. (See also page 1049.)

■ **Parametrizations of growth.** The idea that different objects—say different human bodies or faces—can be related by changing a small number of geometrical parameters was used by artists such as Albrecht Dürer in the 1500s, and may have been known to architects and others in antiquity. (In modern times this idea is associated for example with the notion that just a few measurements are sufficient to specify the fitting of clothes.) D'Arcy Thompson in 1917 suggested that shapes in many different species could also be related in this way. In the case of shells and horns he gave a fairly precise analysis, as discussed above. But he also drew various pictures of fishes and skulls, and argued that they were related by deformations of coordinates. Largely from this grew the field of morphometrics, in which the relative positions of features such as eyes or tips of fins are compared in different species. And although statistical significance is reduced by considering only discrete features, some evidence has emerged that different species do indeed have shapes related by changes in fairly small numbers of geometrical parameters. Such changes could be accounted for by changes in growth rates, but it is noteworthy that my results above on branching and folding make it clear that in general changes in growth rates can have much more dramatic effects.

As emphasized by D'Arcy Thompson, even a single organism will change shape if its parts grow at different rates. And in the 1930s and 1940s it became popular to study differential growth, typically under the name of allometry. Exponential growth was usually assumed, and there was much discussion about the details and correctness of this. Practical applications were made to farm animals, and later to changes in facial bone structure during childhood. But despite some work in the past twenty years using models based on fluids, solid mechanics and networks of rigid elements, much about differential growth remains unclear. (A better approach may be one similar to general relativity.)

▪ **Schemes for growth.** After the initial embryonic stage, many general features of the growth of different types of organisms can be viewed as consequences of the nature of the elements that make the organisms rigid. In plants, as we have discussed, essentially all cells have rigid cellulose walls. In vertebrate animals, rigidity comes mainly from bones that are internal to the organism. In arthropods and some other invertebrates, an exoskeleton is typically the main source of rigidity. Growth in such organisms usually then proceeds by adding soft tissue on the inside, then periodically moulting the exoskeleton. In a first approximation, the mechanical pressure of internal tissue will typically make the shape of the exoskeleton form an approximate minimal surface.

▪ **Tumors.** In both plants and animals tumors seem to grow mainly by fairly random addition of cells to their surface—much as in the aggregation models shown on page 332.

▪ **Pollen.** The grains of pollen produced by different species of plants have a remarkable range of different forms. Produced in groups of four, each grain is effectively a single cell (with two nuclei) between a few and few hundred microns across. At an overall level most grains seem to have regular polyhedral shapes, though often with bulges or dents. Perhaps such forms arise through grains effectively being made with small numbers of roughly spherical elements being either as tightly or loosely packed as possible. The outer walls of pollen grains are often covered with a certain density of tiny columns that can form spikes, or can have plates on top that can form cross-linkages and can join together to appear as patches.

▪ **Radiolarians.** The silicate skeletons of single-cell plankton organisms such as radiolarians and diatoms have been used for well over a century as examples of complex microscopic forms in biology. (See page 385.) Most likely their overall shapes are determined before they harden through minimization of area by surface tension. Their pores and cross-linkages presumably reflect packings of many roughly spherical elements on the surface during formation (as seen in the mid-1990s in aluminophosphates).

▪ **Self-assembly.** Some growth—particularly at a microscopic level—seems to be based on objects with particular shapes or affinities sticking together only in specific ways—much as in the systems based on constraints discussed on page 210 (and especially the network constraint systems of page 483). (See also page 1193.)

▪ **Animal behavior.** Simple repetitive behavior is common, as in circadian and brain rhythms, as well as peristalsis and walking. (In a millipede there are, for example, typically just two modes of locomotion, both simple, involving opposite legs moving either together or oppositely.) Many structures built by animals have repetitive forms, as in beehives and spider webs; the more complex structures made for example by termites can perhaps be understood in terms of generalized aggregation systems (see page 978). (Typical models involve the notion of stigmergy: that elements are added at a particular point based only on features immediately around them; see also page 1184.) Nested patterns may occur in flocks of birds such as geese. Fairly regular nested space-filling curves are sometimes seen in the eating paths of caterpillars. Apparent randomness is common in physiological processes such as twitchings of muscles and microscopic eye motions, as well as in random walks executed during foraging. My suspicion is that just as there appear to be small collections of cells—so-called central pattern generators—that generate repetitive behavior, so also there will turn out to be small collections of cells that generate intrinsically random behavior.

Biological Pigmentation Patterns

▪ **Collecting shells.** The shells I show in this section are mostly from my fairly small personal collection, obtained at shops and markets around the world. (A few of the ones on page 416 are from the Field Museum.) The vast majority of shells on typical beaches do not have especially elaborate patterns. The Philippines are the largest current source of collectible shells: when molluscs intended as food are caught in nets interesting-looking shells are sometimes picked out before being discarded. Shell collecting as a hobby probably had its greatest popularity in the late 1700s and 1800s. In recent times one reason for studying animals that live in cone shells is that they produce potent neurotoxins that show promise as pain-control drugs.

▪ **Shell patterns.** The so-called mantle of soft tissue which covers the animal inside the shell is what secretes the shell and produces the pattern on it. Some species deposit material in a highly regular way every day; others seem to do it intermittently over periods of months or years. In many species the outer surface of the shell is covered by a kind of skin known as the periostracum, and in most cases this skin is opaque, thereby obscuring the patterns underneath until long after the animal has died. Note that if one makes a hole in a shell, the pattern is usually quite unaffected, suggesting that the pattern is primarily a consequence of features of the underlying mantle. In addition, patterns are often divided into three or four large bands, presumably in correspondence with features of the anatomy of the mantle. Sometimes physical ridges exist on shells in correspondence with their

pigmentation patterns. It is not clear whether multiple kinds of shell patterns can occur within one species, or whether they are always associated with genetically different species.

■ **Cowries.** In cowries the outside of the shell is covered by the mantle of the animal. The patterns on the shell typically involve spots, and are typical of those obtained from 2D cellular automata of the kind shown on page 428. The mantle is normally in two parts; the boundary between them shows up as a discontinuity in the shell pattern.

■ **History.** Elaborate patterns on shells have been noticed since antiquity, and have featured in a number of well-known works of art and literature. Since the late 1600s they have also been extensively used in classifying molluscs. But almost no efforts to understand the origins of such patterns seem to ever have been made. One study was done in 1969 by Conrad Waddington and Russell Cowe in which patterns on one particular kind of shell were reproduced by a specific computer simulation based on the idea of diverging waves of pigment. In 1982 I noticed that the patterns I had generated with 1D cellular automata looked remarkably similar to patterns on shells. I used this quite widely as an illustration of how cellular automata might be relevant to modelling natural systems. And I also made some efforts to do actual biological experiments, but I gave up when it seemed that the species of molluscs I wanted to study were difficult, if not impossible, to keep in captivity. Following my work, various other studies of shell patterns were done. Bard Ermentrout, John Campbell and George Oster constructed a model based on the idea that pigment-producing cells might act like nerve cells with a certain degree of memory. And Hans Meinhardt has constructed progressively more elaborate models based on reaction-diffusion equations.

■ **Page 426 · Animals shown.** Flatworm, cuttlefish, honeycomb moray, spotted moray, foureye butterfly fish; emperor angelfish, suckermouth catfish, ornate cowfish, clown triggerfish, poison-dart frog; ornate horned frog, marbled salamander, spiny softshell, gila monster, ball python; gray-banded kingsnake, guinea fowl, peacock, ring-tailed lemur, panda; cheetah, ocelot, leopard, tiger, spotted hyena; western spotted skunk, civet, zebra, brazilian tapir, giraffe.

■ **Animal coloration.** Coloration can arise either directly through the presence of visible colored cells such as those in freckles, or indirectly by virtue of cells such as hair follicles imparting pigments to the non-living elements such as fur, feathers and scales that grow from them. In many cases such elements are arranged in a highly regular way, often in a repetitive hexagonal pattern. Evolutionary optimization is often used to explain observed pigmentation patterns—with varying degrees of success. The notion that for example the stripes of a zebra are for camouflage may at first seem implausible, but there is some evidence that dramatic stripes do make it harder for a predator to recognize the overall shape of the zebra. Many of the pigments used by animals are by-products of metabolism, suggesting that at least at first pigmentation patterns were probably often incidental to the operation of the animal.

■ **Page 427 · Implementation.** Given a 2D array of values a and a list of weights w, each step in the evolution of the system corresponds to

```
WeightedStep[w_List, a_] := Map[If[# > 0, 1, 0] &,
    Sum[w[[1 + i]] Apply[Plus, Map[RotateLeft[a, #] &,
        Layer[i]]], {i, 0, Length[w] - 1}], {2}]
Layer[n_] := Layer[n] = Select[Flatten[Table[{i, j},
    {i, -n, n}, {j, -n, n}], 1], MemberQ[#, n | -n] &]
```

■ **Features of the model.** The model is a totalistic 2D cellular automaton, as discussed on page 927. It shows class 2 behavior in which information propagates only over limited distances, so that except when the total size of the system is comparable to the range of the rule, boundary conditions are not crucial.

Similar models have been considered before. In the early 1950s (see below) Alan Turing used a model which effectively differed mainly in having continuous color levels. In 1979 Nicholas Swindale constructed a model with discrete levels to investigate ocular dominance stripes in the brain (see below). And following my work on cellular automata in the early 1980s, David Young in 1984 considered a model even more similar to the one I use here.

There are simple cellular automata—such as 8-neighbor outer totalistic code 196623—which eventually yield maze-like patterns even when started from simple initial conditions. The rule on page 336 gives dappled patterns with progressively larger spots.

■ **Reaction-diffusion processes.** The cellular automaton in the main text can be viewed as a discrete idealization of a reaction-diffusion process. The notion that diffusion might be important in embryo development had been suggested in the early 1900s (see page 1004), but it was only in 1952 that Alan Turing showed how it could lead to the formation of definite patterns. Diffusion of a single chemical always tends to smooth out distributions of concentration. But Turing pointed out that with two chemicals in which each can be produced from the other it is possible for separated regions to develop. If $c = \{u[t, x], v[t, x]\}$ is a vector of chemical concentrations, then for suitable values of parameters even the standard linear diffusion equation $\partial_t c == d \,.\, \partial_{xx} c + m \,.\, c$

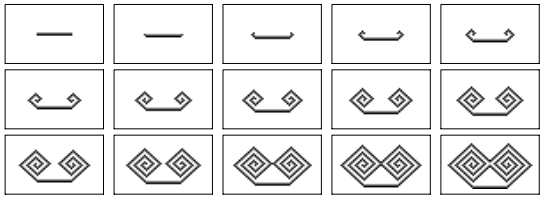
can exhibit an instability which causes disturbances with certain spatial wavelengths to grow (compare page 988). In his 1952 paper Turing used a finite difference approximation to a pair of diffusion equations to show that starting from a random distribution of concentration values dappled regions could develop in which one or the other chemical was dominant. With purely linear equations, any instability will always eventually lead to infinite concentrations, but Turing noted that this could be avoided by using realistic nonlinear chemical rate equations. In the couple of years before his death in 1954, Turing appears to have tried to simulate such nonlinear equations on an early digital computer, but my cursory efforts to understand his programs—written as they are in a 32-character machine code—were not successful.

Following Turing's work, the fact that simple reaction-diffusion equations can yield spatially inhomogeneous patterns has been rediscovered—with varying degrees of independence—many times. In the early 1970s Ilya Prigogine termed the patterns dissipative structures. And in the mid-1970s, Hermann Haken considered the phenomenon a cornerstone of what he called synergetics.

Many detailed mathematical analyses of linear reaction-diffusion equations have been done since the 1970s; numerical solutions to linear and occasionally nonlinear such equations have also often been found, and in recent years explicit pictures of patterns—rather than just curves of related functions—have commonly been generated. In the context of biological pigmentation patterns detailed studies have been done particularly by Hans Meinhardt and James Murray.

▪ **Scales of patterns.** The visual appearance of a pattern on an actual animal depends greatly on the scale of the pattern relative to the whole animal. Pandas and anteaters, for example, typically have just a few regions of different color, while other animals can have hundreds of regions. Studies based on linear reaction-diffusion equations sometimes assume that patterns correspond to stationary modes of the equations, which inevitably depend greatly on boundary conditions. But in more realistic models patterns emerge from long time behavior with generic initial conditions, making boundary conditions—and effects such as changes in them associated with growth of an embryo—much less important.

▪ **Excitable media.** In many physical situations effects become decreasingly important as they propagate further away. But in active or excitable media such as heart, muscle and nerve tissue an effect can maintain its magnitude as it propagates, leading to the formation of a variety of spatial structures. An early model of such media was constructed in 1946 by Norbert Wiener and Arturo Rosenblueth, based on a discrete array of continuous elements. Models with discrete elements were already considered in the 1960s, and in 1977 James Greenberg and Stuart Hastings introduced a simple 2D cellular automaton with three colors. The pictures below show what is probably the most complex feature of this cellular automaton and related systems: the formation of spiral waves. Such spiral waves were studied in 2D and 3D in the 1970s and 1980s, particularly by Arthur Winfree and others; they are fairly easy to observe in both inorganic chemical reactions (see below) and slime mold colonies.



▪ **Examples in chemistry.** Overall concentrations in chemical reactions can be described by nonlinear ordinary differential equations. Reactions with oscillatory behavior were predicted by Alfred Lotka in 1910 and observed experimentally by William Bray in 1917, but for some reason they were not further investigated at that time. An example was found experimentally by Boris Belousov in 1951 and extensive investigations of it were begun by Anatol Zhabotinsky around 1960. In the early 1970s spiral waves were seen in the spatial distribution of concentrations in this reaction, and by the end of the 1970s images of such waves were commonly used as icons of the somewhat ill-defined notion of self-organization.

▪ **Maze-like patterns.** Maze-like patterns occur in several quite different kinds of systems. Cases in which the underlying mechanism is probably similar to that discussed in the main text include brain coral, large-scale vegetation bands seen in tropical areas, patterns of sand dunes, patterns in pre-turbulent fluid convection, and ocular dominance stripes consisting of regions of brain tissue that get marked when different dye is introduced into nerves from left and right eyes. Cases in which the underlying mechanism is probably more associated with folding of fixed amounts of material include human fingerprint patterns and patterns in ferrofluids consisting of suspensions of magnetic particles.

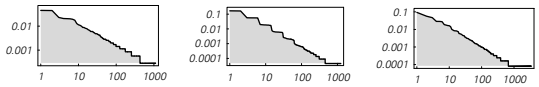
▪ **Origins of randomness.** The model in the main text assumes that randomness enters through initial conditions. If the two parts of a single animal—say opposite wings on a butterfly—

form together, then these initial conditions can be expected to be the same. But usually even the two sides of a single animal are never physically together, and they normally end up having quite uncorrelated random features. In cases such as fingerprints and zebra stripes there is some correlation between different sides, suggesting an intrinsic component to the randomness that occurs. (The fingerprints of identical twins are typically similar but not identical; iris patterns are quite different.) Note that at least sometimes random initial patterns are formed by cells that have the same type, but different lineages—as in cells expressing genes from the two different X chromosomes in a female animal such as a typical tortoiseshell cat. (In general, quite a few traits—particularly related to aging—can show significant variation in strains of organisms that are genetically identical.)

Financial Systems

■ **Laws of human behavior.** Over the past century there have been a fair number of quantitative laws proposed for features of human behavior. Some are presumably a direct reflection of human biological construction. Thus for example, Weber's law that the perceived strength of a stimulus tends to vary logarithmically with its actual strength seems likely to be related to the electrochemistry of nerve cells. Of laws for more complicated cognitive or social phenomena the vast majority are statistical in nature. And of those that withstand scrutiny, most in my experience turn out to be transformed versions of statements that some quantity or another can be approximated by perfect randomness. Gaussian distributions typically arise when measurements involve sums of random quantities; other common distributions are obtained from products or other simple combinations of random quantities, or from the results of simple processes based on random quantities. Exponential distributions (as seen, for example, in learning curves) and power-law distributions (as in Zipf's law below) are both, for example, very easy to obtain. (Note that particularly in economics there are also various laws derived from calculus and game theory that are viewed as being quite successful, and are not fundamentally statistical.)

■ **Zipf's law.** To a fairly good approximation the n^{th} most common word in a large sample of English text occurs with frequency $1/n$, as illustrated in the first picture below. This fact was first noticed around the end of the 1800s, and was attributed in 1949 by George Zipf to a general, though vague, Principle of Least Effort for human behavior. I suspect that in fact the law has a rather simple probabilistic origin. Consider generating a long piece of text by picking at random from k letters and a space. Now collect and rank all the "words" delimited by spaces that are formed. When $k = 1$, the n^{th} most common word will have frequency c^{-n}. But when $k \geq 2$, it turns out that the n^{th} most common word will have a frequency that approximates c/n. If all k letters have equal probabilities, there will be many words with equal frequency, so the distribution will contain steps, as in the second picture below. If the k letters have non-commensurate probabilities, then a smooth distribution is obtained, as in the third picture. If all letter probabilities are equal, then words will simply be ranked by length, with all k^m words of length m occurring with frequency p^m. The normalization of probabilities then implies $p = 1/(2k)$, and since the word at rank roughly k^m then has probability $1/(2k)^m$, Zipf's law follows.



■ **Motion of people and cars.** To a first approximation crowds of people seem to show aggregate fluid-like behavior similar to what is seen in gases. Fronts of people—as occur in riots or infantry battles—seem to show instabilities perhaps analogous to those in fluids. Road traffic that is constrained to travel along a line exhibits stop-start instabilities when its overall rate is reduced, say by an obstruction. This appears to be a consequence of the delay before one driver responds to changes in speed of cars in front of them. Fairly accurate cellular automaton models of this phenomenon were developed in the early 1990s.

■ **Growth of cities.** In the absence of geographical constraints, such as terrain or oceans, cities typically have patchy, irregular, shapes. At first an aggregation system (see page 331) might seem to be an obvious model for their growth: each new development gets added to the exterior of the city at a random position. But actual cities look much more irregular. Most likely the reason is that embedded within the cities is a network of transportation routes, and these tend to have a tree- or vein-like structure (though not necessarily with a single center)—with major freeways etc. as trunks. The result of following this structure is to produce a much more irregular boundary.

■ **Randomness in markets.** After the somewhat tricky process of correcting for overall trends, empirical price data from a wide range of markets seem to a first approximation to follow random walks and thus to exhibit Gaussian fluctuations, as noted by Louis Bachelier in 1900. However, particularly on timescales less than a day, it has in the past decade become clear that, as suggested by Benoit Mandelbrot in the early 1960s, large price fluctuations are significantly more common than a Gaussian distribution would imply.

Such an effect is easy to model with the approach used in the main text if different entities interact in clumps or herds—which can be forced if they are connected in a hierarchical network rather than just a line.

The observed standard deviation of a price—or essentially so-called volatility or beta—can be considered as a measure of the risk of fluctuations in that price. The Capital Asset Pricing Model proposed in the early 1960s suggested that average rates of price increases should be proportional to such variances. And the Black-Scholes model from 1973 implies that prices of suitably constructed options should depend in a sense only on such variances. Over the past decade various corrections to this model have been developed based on non-Gaussian distributions of prices.

▪ **Speculative markets.** Cases of markets that seem to operate almost completely independent of objective value have occurred many times in economic history, particularly in connection with innovations in technology or finance. Examples range from tulip bulbs in the mid-1630s to railroads in the mid-1800s to internet businesses in the late 1990s. (Note however that in any particular case it can be claimed that certain speculation was rational, even if it did not work out—but usually it is difficult to get convincing evidence for this, and often effects are obscured by generalized money supply or bankruptcy issues.)

▪ **Properties of markets.** Issues of how averaging is done and how irrelevant trends are removed turn out to make unequivocal tests of almost any quantitative hypothesis about prices essentially impossible. The rational expectations theory that prices reflect discounted future earnings has for example been subjected to many empirical tests, but has never been convincingly proved or disproved.

▪ **Efficient markets.** In its strong form the so-called Efficient Market Hypothesis states that prices immediately adjust to reflect all possible information, so that knowing a particular piece of information can never be used to make a profit. It is now widely recognized—even in academia—that this hypothesis is a fairly poor representation of reality.

▪ **Details of trading.** Cynics might suggest that much of the randomness in practical markets is associated with details of trading. For much of the money actually made from markets on an ongoing basis comes from commissions on trades. And if prices quickly settled down to their final values, fewer trades would tend to be made. (Different entities would nevertheless still often need liquidity at different times.)

▪ **Models of markets.** When serious economic theory began in the 1700s arguments tended to be based purely on common sense. But with the work of Léon Walras in the 1870s mathematical models began to become popular. In the early 1900s, common sense again for a while became dominant. But particularly with the development of game theory in the 1940s the notion became established, at least in theoretical economics, that prices represent equilibrium points whose properties can be derived mathematically from requirements of optimality. In practical trading, partly as an outgrowth of theories of business cycles, there had emerged all sorts of elaborate so-called technical analysis in which patterns of price movements were supposed—often on the basis of almost mystical theories—to be indicators of future behavior. In the late 1970s, particularly after the work of Fischer Black and Myron Scholes on options pricing, new models of markets based on methods from statistical physics began to be used, but in these models randomness was taken purely as an assumption. In another direction, it was noticed that dynamic versions of game theory could yield iterated maps and ordinary differential equations which would lead to chaotic behavior in prices, but connections with randomness in actual markets were not established. By the mid-1980s, however, it began to be clear that the whole game-theoretical idea of thinking of markets as collections of rational entities that optimize their positions on the basis of complete information was quite inadequate. Some attempts were made to extend traditional mathematical models, and various highly theoretical analyses were done based on treating entities in the market as universal computers. But by the end of the 1980s, the idea had emerged of doing explicit computer simulations with entities in the market represented by practical programs. (See also page 1105.) Often these programs used fairly sophisticated algorithms intended to mimic human traders, but in competitions between programs simpler algorithms have never seemed to be at much of a disadvantage. The model in the main text is in a sense an ultimate idealization along these lines. It follows a sequence of efforts that I have made since the mid-1980s—though have never considered very satisfactory—to find minimal but accurate models of financial processes.

Fundamental Physics

The Notion of Reversibility

■ **Page 437 · Testing for reversibility.** To show that a cellular automaton is reversible it is sufficient to check that all configurations consisting of repetitions of different blocks have different successors. This can be done for blocks up to length n in a 1D cellular automaton with k colors using

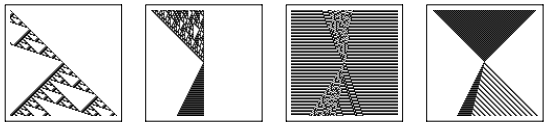
```
ReversibleQ[rule_, k_, n_] := Catch[Do[
  If[Length[Union[Table[CAStep[rule, IntegerDigits[i, k, m]],
    {i, 0, k^m - 1}]]] ≠ k^m, Throw[False]], {m, n}]; True]
```

For $k = 2$, $r = 1$ it turns out that it suffices to test only up to $n = 4$ (128 out of the 256 rules fail at $n = 1$, 64 at $n = 2$, 44 at $n = 3$ and 14 at $n = 4$); for $k = 2$, $r = 2$ it suffices to test up to $n = 15$, and for $k = 3$, $r = 1$, up to $n = 9$. But although these results suggest that in general it should suffice to test only up to $n = k^{2r}$, all that has so far been rigorously proved is that $n = k^{2r} (k^{2r} - 1) + 2r + 1$ (or $n = 15$ for $k = 2$, $r = 1$) is sufficient.

For 2D cellular automata an analogous procedure can in principle be used, though there is no upper limit on the size of blocks that need to be tested, and in fact the question of whether a particular rule is reversible is directly equivalent to the tiling problem discussed on page 213 (compare page 942), and is thus formally undecidable.

■ **Numbers of reversible rules.** For $k = 2$, $r = 1$, there are 6 reversible rules, as shown on page 436. For $k = 2$, $r = 2$ there are 62 reversible rules, in 20 families inequivalent under symmetries, out of a total of 2^{32} or about 4 billion possible rules. For $k = 3$, $r = 1$ there are 1800 reversible rules, in 172 families. For $k = 4$, $r = 1$, some of the reversible rules can be constructed from the second-order cellular automata below. Note that for any k and r, no non-trivial totalistic rule can ever be reversible.

■ **Inverse rules.** Some reversible rules are self-inverse, so that applying the same rule twice yields the identity. Other rules come in distinct pairs. Most often a rule that involves r neighbors has an inverse that also involves at most r neighbors. But for both $k = 2$, $r = 2$ and $k = 3$, $r = 1$ there turn out to be reversible rules whose inverses involve larger numbers of neighbors. For any given rule one can define the neighborhood size s to be the largest block of cells that is ever needed to determine the color of a single new cell. In general $s \le 2r + 1$, and for a simple identity or shift rule, $s = 1$. For $k = 2$, $r = 1$, it then turns out that all the reversible rules and their inverses have $s = 1$. For $k = 2$, $r = 2$, the reversible rules have values of s from 1 to 5, but their inverses have values $\bar{s}$ from 1 to 6. There are only 8 rules (the inequivalent ones being 16740555 and 3327051468) where $\bar{s} > s$, and in each case $\bar{s} = 6$ while $s = 5$. For $k = 3$, $r = 1$, there are a total of 936 rules with this property: 576, 216 and 144 with $\bar{s} = 4$, 5 and 6, and in all cases $s = 3$. Examples with $\bar{s} = 3$, 4, 5 and 6 are shown below. For arbitrary k and r, it is not clear what the maximum $\bar{s}$ can be; the only bound rigorously established so far is $\bar{s} \le r + 1/2\, k^{2r+1} (k^{2r} - 1)$.



2828556973047 3762560660157 538556225233 3066231781977

■ **Surjectivity and injectivity.** See page 959.

■ **Directional reversibility.** Even if successive time steps in the evolution of a cellular automaton do not correspond to an injective map, it is still possible to get an injective map by looking at successive lines at some angle in the spacetime evolution of the system. Examples where this works include the surjective rules 30 and 90.

■ **Page 437 · Second-order cellular automata.** Second-order elementary rules can be implemented using

```
CA2EvolveList[rule_List, {a_List, b_List}, t_Integer] :=
  Map[First, NestList[CA2Step[rule, #] &, {a, b}, t]]
CA2Step[rule_List, {a_, b_}] := {b, Mod[a + rule[[
    8 - (RotateLeft[b] + 2 (b + 2 RotateRight[b]))]], 2]}
```

where *rule* is obtained from the rule number using *IntegerDigits[n, 2, 8]*.

The combination *Drop[list, -1] + 2 Drop[list, 1]* of the result from *CA2EvolveList* corresponds to evolution according to a first-order $k = 4$, $r = 1$ rule.

■ **History.** The concept of getting reversibility in a cellular automaton by having a second-order rule was apparently first suggested by Edward Fredkin around 1970 in the context of 2D systems—on the basis of an analogy with second-order differential equations in physics. Similar ideas had appeared in numerical analysis in the 1960s in connection with so-called symmetric or self-adjoint discrete approximations to differential equations.

■ **Page 438 · Properties.** The pattern from rule 67R with simple initial conditions grows irregularly, at an average rate of about 1 cell every 5 steps. The right-hand side of the pattern from rule 173R consists three triangles that repeat progressively larger at steps of the form $2(9^s - 1)$. Rule 90R has the property that of the diamond of cells at relative positions *{{-n, 0}, {0, -n}, {n, 0}, {0, n}}* it is always true for any n that an even number are black.

■ **Page 439 · Properties.** The initial conditions used here have a single black cell on two successive initial steps. For rule 150R, however, there is no black cell on the first initial step. The pattern generated by rule 150R has fractal dimension $\text{Log}[2, 3 + \sqrt{17}] - 1$ or about 1.83. In rule 154R, each diagonal stripe is followed by at least one 0; otherwise, the positions of the stripes appear to be quite random, with a density around 0.44.

■ **Generalized additive rules.** Additive cellular automata of the kind discussed on page 952 can be generalized by allowing the new value of each cell to be obtained from combinations of cells on s previous steps. For rule 90 the combination c can be specified as *{{1, 0, 1}}*, while for rule 150R it can be specified as *{{0, 1, 0}, {1, 1, 1}}*. All generalized additive rules ultimately yield nested patterns. Starting with a list of the initial conditions for s steps, the configurations for the next s steps are given by

```
Append[Rest[list],
  Map[Mod[Apply[Plus, Flatten[c#]], 2] &, Transpose[
    Table[RotateLeft[list, {0, i}], {i, -r, r}], {3, 2, 1}]]]
```

where $r = (\text{Length}[\text{First}[c]] - 1)/2$.

Just as for ordinary additive rules on page 1091, an algebraic analysis for generalized additive rules can be given. The objects that appear are solutions to linear recurrences of order s, and in general involve s^{th} roots. For rule 150R, the configuration at step t as shown in the picture on page 439 is given by $(u^t - v^t)/\text{Sqrt}[4 + h^2]$, where $\{u, v\} = z /. \text{Solve}[z^2 == h z + 1]$ and $h = 1/x + 1 + x$. (See also page 1078.)

■ **Page 440 · Rule 37R.** Complicated structures are fairly easy to get with this rule. The initial condition *{1, 0, 1}* with all cells 0 on the previous step yields a structure that repeats but only every 666 steps. The initial condition *{{0, 1, 1}, {1, 0, 0}}* yields a pattern that grows sporadically for 3774 steps, then breaks into two repetitive structures. The typical background repeats every 3 steps.

■ **Classification of reversible rules.** In a reversible system it is possible with suitable initial conditions to get absolutely any arrangement of cells to appear at any step. Despite this, however, the overall spacetime pattern of cells is not arbitrary, but is instead determined by the underlying rules. If one starts with completely random initial conditions then class 2 and class 3 behavior are often seen. Class 1 behavior can never occur in a reversible system. Class 4 behavior can occur, as in rule 37R, but is typically obvious only if one starts say with a low density of black cells.

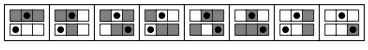
For arbitrary rules, difference patterns of the kind shown on page 250 can get both larger and smaller. In a reversible rule, such patterns can grow and shrink, but can never die out completely.

■ **Emergence of reversibility.** Once on an attractor, any system—even if it does not have reversible underlying rules—must in some sense show approximate reversibility. (Compare page 959.)

■ **Other reversible systems.** Reversible examples can be found of essentially all the types of systems discussed in this book. Reversible mobile automata can for instance be constructed using

```
Table[(IntegerDigits[i, 2, 3] → If[First[#] == 0, {#, -1},
    {Reverse[#], 1}] &)[IntegerDigits[perm[[i]], 2, 3]], {i, 8}]
```

where *perm* is an element of *Permutations[Range[8]]*. An example that exhibits complex behavior is:



Systems based on numbers are typically reversible whenever the mathematical operations they involve are invertible. Thus, for example, the system on page 121 based on successive multiplication by 3/2 is reversible by using division by 3/2. Page 905 gives another example of a reversible system based on numbers.

Multiway systems are reversible whenever both $a \to b$ and $b \to a$ are present as rules, so that the system corresponds mathematically to a semigroup. (See page 938.)

■ **Reversible computation.** Typical practical computers—and computer languages—are not even close to reversible: many inputs can lead to the same output, and there is no unique

way to undo the steps of a computation. But despite early confusion (see page 1020), it has been known since at least the 1970s that there is nothing in principle which prevents computation from being reversible. And indeed—just like with the cellular automata in this section—most of the systems in Chapter 11 that exhibit universal computation can readily be made reversible with only slight overhead.

Irreversibility and the Second Law of Thermodynamics

■ **Time reversal invariance.** The reversibility of the laws of physics implies that given the state of a physical system at a particular time, it is always possibly to work out uniquely both its future and its past. Time reversal invariance would further imply that the rules for going in each direction should be identical. To a very good approximation this appears to be true, but it turns out that in certain esoteric particle physics processes small deviations have been found. In particular, it was discovered in 1964 that the decay of the K^0 particle violated time reversal invariance at the level of about one part in a thousand. In current theories, this effect is not attributed any particularly fundamental origin, and is just assumed to be associated with the arbitrary setting of certain parameters. K^0 decay was for a long time the only example of time reversal violation that had explicitly been seen, although recently examples in B particle decays have probably also been seen. It also turns out that the only current viable theories of the apparent preponderance of matter over antimatter in the universe are based on the idea that a small amount of time reversal violation occurred in the decays of certain very massive particles in the very early universe.

The basic formalism used for particle physics assumes not only reversibility, but also so-called CPT invariance. This means that same rules should apply if one not only reverses the direction of time (T), but also simultaneously inverts all spatial coordinates (P) and conjugates all charges (C), replacing particles by antiparticles. In a certain mathematical sense, CPT invariance can be viewed as a generalization of relativistic invariance: with a speed faster than light, something close to an ordinary relativistic transformation is a CPT transformation.

Originally it was assumed that C, P and T would all separately be invariances, as they are in classical mechanics. But in 1957 it was discovered that in radioactive beta decay, C and P are in a sense each maximally violated: among other things, the correlation between spin and motion direction is exactly opposite for neutrinos and for antineutrinos that are emitted. Despite this, it was still assumed that CP and T would be true invariances. But in 1964 these too were found to be violated. Starting with a pure beam of K^0 particles, it turns out that quantum mechanical mixing processes lead after about 10^{-8} seconds to a certain mixture of $\overline{K}^0$ particles—the antiparticles of the K^0. And what effectively happens is that the amount of mixing differs by about 0.1% in the positive and negative time directions. (What is actually observed is a small probability for the long-lived component of a K^0 beam to decay into two rather than three pions. Some analysis is required to connect this with T violation.) Particle physics experiments so far support exact CPT invariance. Simple models of gravity potentially suggest CPT violation (as a consequence of deviations from pure special relativistic invariance), but such effects tend to disappear when the models are refined.

■ **History of thermodynamics.** Basic physical notions of heat and temperature were established in the 1600s, and scientists of the time appear to have thought correctly that heat is associated with the motion of microscopic constituents of matter. But in the 1700s it became widely believed that heat was instead a separate fluid-like substance. Experiments by James Joule and others in the 1840s put this in doubt, and finally in the 1850s it became accepted that heat is in fact a form of energy. The relation between heat and energy was important for the development of steam engines, and in 1824 Sadi Carnot had captured some of the ideas of thermodynamics in his discussion of the efficiency of an idealized engine. Around 1850 Rudolf Clausius and William Thomson (Kelvin) stated both the First Law—that total energy is conserved—and the Second Law of Thermodynamics. The Second Law was originally formulated in terms of the fact that heat does not spontaneously flow from a colder body to a hotter. Other formulations followed quickly, and Kelvin in particular understood some of the law's general implications. The idea that gases consist of molecules in motion had been discussed in some detail by Daniel Bernoulli in 1738, but had fallen out of favor, and was revived by Clausius in 1857. Following this, James Clerk Maxwell in 1860 derived from the mechanics of individual molecular collisions the expected distribution of molecular speeds in a gas. Over the next several years the kinetic theory of gases developed rapidly, and many macroscopic properties of gases in equilibrium were computed. In 1872 Ludwig Boltzmann constructed an equation that he thought could describe the detailed time development of a gas, whether in equilibrium or not. In the 1860s Clausius had introduced entropy as a ratio of heat to temperature, and had stated the Second Law in terms of the increase of this quantity. Boltzmann then showed that his

equation implied the so-called H Theorem, which states that a quantity equal to entropy in equilibrium must always increase with time. At first, it seemed that Boltzmann had successfully proved the Second Law. But then it was noticed that since molecular collisions were assumed reversible, his derivation could be run in reverse, and would then imply the opposite of the Second Law. Much later it was realized that Boltzmann's original equation implicitly assumed that molecules are uncorrelated before each collision, but not afterwards, thereby introducing a fundamental asymmetry in time. Early in the 1870s Maxwell and Kelvin appear to have already understood that the Second Law could not formally be derived from microscopic physics, but must somehow be a consequence of human inability to track large numbers of molecules. In responding to objections concerning reversibility Boltzmann realized around 1876 that in a gas there are many more states that seem random than seem orderly. This realization led him to argue that entropy must be proportional to the logarithm of the number of possible states of a system, and to formulate ideas about ergodicity. The statistical mechanics of systems of particles was put in a more general context by Willard Gibbs, beginning around 1900. Gibbs introduced the notion of an ensemble—a collection of many possible states of a system, each assigned a certain probability. He argued that if the time evolution of a single state were to visit all other states in the ensemble—the so-called ergodic hypothesis—then averaged over a sufficiently long time a single state would behave in a way that was typical of the ensemble. Gibbs also gave qualitative arguments that entropy would increase if it were measured in a "coarse-grained" way in which nearby states were not distinguished. In the early 1900s the development of thermodynamics was largely overshadowed by quantum theory and little fundamental work was done on it. Nevertheless, by the 1930s, the Second Law had somehow come to be generally regarded as a principle of physics whose foundations should be questioned only as a curiosity. Despite neglect in physics, however, ergodic theory became an active area of pure mathematics, and from the 1920s to the 1960s properties related to ergodicity were established for many kinds of simple systems. When electronic computers became available in the 1950s, Enrico Fermi and others began to investigate the ergodic properties of nonlinear systems of springs. But they ended up concentrating on recurrence phenomena related to solitons, and not looking at general questions related to the Second Law. Much the same happened in the 1960s, when the first simulations of hard sphere gases were led to concentrate on the specific phenomenon of long-time tails. And by the 1970s, computer experiments were mostly oriented towards ordinary differential equations and strange attractors, rather than towards systems with large numbers of components, to which the Second Law might apply. Starting in the 1950s, it was recognized that entropy is simply the negative of the information quantity introduced in the 1940s by Claude Shannon. Following statements by John von Neumann, it was thought that any computational process must necessarily increase entropy, but by the early 1970s, notably with work by Charles Bennett, it became accepted that this is not so (see page 1018), laying some early groundwork for relating computational and thermodynamic ideas.

▪ **Current thinking on the Second Law.** The vast majority of current physics textbooks imply that the Second Law is well established, though with surprising regularity they say that detailed arguments for it are beyond their scope. More specialized articles tend to admit that the origins of the Second Law remain mysterious. Most ultimately attribute its validity to unknown constraints on initial conditions or measurements, though some appeal to external perturbations, to cosmology or to unknown features of quantum mechanics.

An argument for the Second Law from around 1900, still reproduced in many textbooks, is that if a system is ergodic then it will visit all its possible states, and the vast majority of these will look random. But only very special kinds of systems are in fact ergodic, and even in such systems, the time necessary to visit a significant fraction of all possible states is astronomically long. Another argument for the Second Law, arising from work in the 1930s and 1940s, particularly on systems of hard spheres, is based on the notion of instability with respect to small changes in initial conditions. The argument suffers however from the same difficulties as the ones for chaos theory discussed in Chapter 6 and does not in the end explain in any real way the origins of randomness, or the observed validity of the Second Law.

With the Second Law accepted as a general principle, there is confusion about why systems in nature have not all dissipated into complete randomness. And often the rather absurd claim is made that all the order we see in the universe must just be a fluctuation—leaving little explanatory power for principles such as the Second Law.

▪ **My explanation of the Second Law.** What I say in this book is not incompatible with much of what has been said about the Second Law before; it is simply that I make more definite some key points that have been left vague before. In particular, I use notions of computation to specify what kinds of initial conditions can reasonably be prepared, and what kinds of measurements can reasonably be made. In a sense

what I do is just to require that the operation of coarse graining correspond to a computation that is less sophisticated than the actual evolution of the system being studied. (See also Chapters 10 and 12.)

▪ **Biological systems and Maxwell's demon.** Unlike most physical systems, biological systems typically seem capable of spontaneously organizing themselves. And as a result, even the original statements of the Second Law talked only about "inanimate systems". In the mid-1860s James Clerk Maxwell then suggested that a demon operating at a microscopic level could reduce the randomness of a system such as a gas by intelligently controlling the motion of molecules. For many years there was considerable confusion about Maxwell's demon. There were arguments that the demon must use a flashlight that generates entropy. And there were extensive demonstrations that actual biological systems reduce their internal entropy only at the cost of increases in the entropy of their environment. But in fact the main point is that if the evolution of the whole system is to be reversible, then the demon must store enough information to reverse its own actions, and this limits how much the demon can do, preventing it, for example, from unscrambling a large system of gas molecules.

▪ **Self-gravitating systems.** The observed existence of structures such as galaxies might lead one to think that any large number of objects subject to mutual gravitational attraction might not follow the Second Law and become randomized, but might instead always form orderly clumps. It is difficult to know, however, what an idealized self-gravitating system would do. For in practice, issues such as the limited size of a galaxy, its overall rotation, and the details of stellar collisions all seem to have major effects on the results obtained. (And it is presumably not feasible to do a small-scale experiment, say in Earth orbit.) There are known to be various instabilities that lead in the direction of clumping and core collapse, but how these weigh against effects such as the transfer of energy into tight binding of small groups of stars is not clear. Small galaxies such as globular clusters that contain less than a million stars seem to exhibit a certain uniformity which suggests a kind of equilibrium. Larger galaxies such as our own that contain perhaps 100 billion stars often have intricate spiral or other structure, whose origin may be associated with gravitational effects, or may be a consequence of detailed processes of star formation and explosion. (There is some evidence that older galaxies of a given size tend to develop more regularities in their structure.) Current theories of the early universe tend to assume that galaxies originally began to form as a result of density fluctuations of non-gravitational origin (and reflected in the cosmic microwave background). But there is evidence that a widespread fractal structure develops—with a correlation function of the form $r^{-1.8}$—in the distribution of stars in our galaxy, galaxies in clusters and clusters in superclusters, perhaps suggesting the existence of general overall laws for self-gravitating systems. (See also page 973.)

As mentioned on page 880, it so happens that my original interest in cellular automata around 1981 developed in part from trying to model the growth of structure in self-gravitating systems. At first I attempted to merge and generalize ideas from traditional areas of mathematical physics, such as kinetic theory, statistical mechanics and field theory. But then, particularly as I began to think about doing explicit computer simulations, I decided to take a different tack and instead to look for the most idealized possible models. And in doing this I quickly came up with cellular automata. But when I started to investigate cellular automata, I discovered some very remarkable phenomena, and I was soon led away from self-gravitating systems, and into the process of developing the much more general science in this book. Over the years, I have occasionally come back to the problem of self-gravitating systems, but I have never succeeded in developing what I consider to be a satisfactory approach to them.

▪ **Cosmology and the Second Law.** In the standard big bang model it is assumed that all matter in the universe was initially in completely random thermal equilibrium. But such equilibrium implies uniformity, and from this it follows that the initial conditions for the gravitational forces in the universe must have been highly regular, resulting in simple overall expansion, rather than random expansion in some places and contraction in others. As I discuss on page 1026 I suspect that in fact the universe as a whole probably had what were ultimately very simple initial conditions, and it is just that the effective rules for the evolution of matter led to rapid randomization, whereas those for gravity did not.

▪ **Alignment of time in the universe.** Evidence from astronomy clearly suggests that the direction of irreversible processes is the same throughout the universe. The reason for this is presumably that all parts of the universe are expanding—with the local consequence that radiation is more often emitted than absorbed, as evidenced by the fact that the night sky is dark. Olbers' paradox asks why one does not see a bright star in every direction in the night sky. The answer is that locally stars are clumped, and light from stars further away is progressively red-shifted to lower energy. Focusing a larger and larger distance away, the light one sees was emitted longer and longer ago. And eventually one sees light emitted when the universe was filled with hot opaque

gas—now red-shifted to become the 2.7K cosmic microwave background.

■ **Poincaré recurrence.** Systems of limited size that contain only discrete elements inevitably repeat their evolution after a sufficiently long time (see page 258). In 1890 Henri Poincaré established the somewhat less obvious fact that even continuous systems also always eventually get at least arbitrarily close to repeating themselves. This discovery led to some confusion in early interpretations of the Second Law, but the huge length of time involved in a Poincaré recurrence makes it completely irrelevant in practice.

■ **Page 446 · Billiards.** The discrete system I consider here is analogous to continuous so-called billiard systems consisting of circular balls in the plane. The simplest case involves one ball bouncing around in a region of a definite shape. In a rectangular region, the position is given by *Mod[a t, {w, h}]* and every point will be visited if the parameters have irrational ratios. In a region that contains fixed circular obstructions, the motion can become sensitively dependent on initial conditions. (This setup is similar to a so-called Lorentz gas.) For a system of balls in a region with cyclic boundaries, a complicated proof due to Yakov Sinai from the 1960s purports to show that every ball eventually visits every point in the region, and that certain simple statistical properties of trajectories are consistent with randomness. (See also page 971.)

■ **Page 449 · Entropy of particles in a box.** The number of possible states of a region of m cells containing q particles is *Binomial[m, q]*. In the large size limit, the logarithm of this can be approximated by $q\,$*Log[m/q]*$/m$.

■ **Page 457 · Periods in rule 37R.** With a system of size n, the maximum possible repetition period is 2^{2^n}. In actuality, however, the periods are considerably shorter. With all cells 0 on one step, and a block of nonzero cells on the next step, the periods are for example: *{1}*: *21*; *{1, 1}*: $3n-8$; *{1, 0, 1}*: *666*; *{1, 1, 1}*: $3n-8$; *{1, 0, 0, 1}*: irregular ($< 24n$; peaks at $6j+1$); *{1, 0, 0, 1, 0, 1}*: irregular ($\lesssim 2^n$; *857727* for $n = 26$; *13705406* for $n = 100$). With completely random initial conditions, there are great fluctuations, but a typical period is around $2^{n/3}$.

Conserved Quantities and Continuum Phenomena

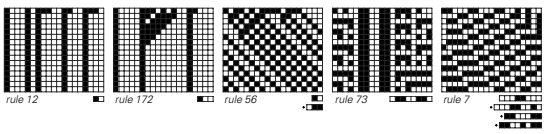
■ **Physics.** The quantities in physics that so far seem to be exactly conserved are: energy, momentum, angular momentum, electric charge, color charge, lepton number (as well as electron number, muon number and τ lepton number) and baryon number.

■ **Implementation.** Whether a k-color cellular automaton with range r conserves total cell value can be determined from

```
Catch[Do[
  (If[Apply[Plus, CAStep[rule, #] - #] ≠ 0, Throw[False]] &)[
   IntegerDigits[i, k, m]], {m, w}, {i, 0, k^m - 1}]; True]
```

where w can be taken to be k^{2r}, and perhaps smaller. Among the 256 elementary cellular automata just 5 conserve total cell value. Among the 2^{32} $k = 2$, $r = 2$ rules 428 do, and of these 2 are symmetric, and 6 are reversible, and all these are just shift and identity rules.

■ **More general conserved quantities.** Some rules conserve not total numbers of cells with given colors, but rather total numbers of blocks of cells with given forms—or combinations of these. The pictures below show the simplest quantities of these kinds that end up being conserved by various elementary rules.



Among the 256 elementary rules, the total numbers that have conserved quantities involving at most blocks of lengths 1 through 10 are *{5, 38, 66, 88, 102, 108, 108, 114, 118, 118}*.

Rules that show complicated behavior usually do not seem to have conserved quantities, and this is true for example of rules 30, 90 and 110, at least up to blocks of length 10.

One can count the number of occurrences of each of the k^b possible blocks of length b in a given state using

```
BC[list_] :=
  With[{z = Map[FromDigits[#, k] &, Partition[list, b, 1, 1]]},
    Map[Count[z, #] &, Range[0, k^b - 1]]]
```

Conserved quantities of the kind discussed here are then of the form $q \cdot$ *BC[a]* where q is some fixed list. A way to find candidates for q is to compute

```
NullSpace[Table[With[{u = Table[Random[Integer,
    {0, k - 1}], {m}]}, BC[CAStep[u]] - BC[u]], {s}]]
```

for progressively larger m and s, and to see what lists continue to appear. For block size b, k^{b-1} lists will always appear as a result of trivial conserved quantities. (With $k = 2$, for $b = 1$, *{1, 1}* represents conservation of the total number of cells, regardless of color, while for $b = 2$, *{1, 1, 1, 1}* represents the same thing, while *{0, 1, -1, 0}* represents the fact that in going along in any state the number of black-to-white transitions must equal the number of white-to-black ones.) If more than k^{b-1} lists appear, however, then some must correspond to genuine non-trivial conserved quantities. To identify any such quantity with certainty, it turns out to be enough to look at the k^{b+2r-1} states where no block of length

$b + 2r - 1$ appears more than once (and perhaps even just some fairly small subset of these).

(See also page 981.)

▪ **Other conserved quantities.** The conserved quantities discussed so far can all be thought of as taking values assigned to blocks of different kinds in a given state and then just adding them up as ordinary numbers. But one can also imagine using other operations to combine such values. Addition modulo n can be handled by inserting $Modulus \to n$ in *NullSpace* in the previous note. And doing this shows for example that rule 150 conserves the total number of black cells modulo 2. But in general not many additional conserved quantities are found in this way. One can also consider combining values of blocks by the multiplication operation in a group—and seeing whether the conjugacy class of the result is conserved.

▪ **PDEs.** In the early 1960s it was discovered that certain nonlinear PDEs support an infinite number of distinct conserved quantities, associated with so-called integrability and the presence of solitons. Systematic methods now exist to find conserved quantities that are given by integrals of polynomials of any given degree in the dependent variables and their derivatives. Most randomly chosen PDEs appear, however, to have no such conserved quantities.

▪ **Local conservation laws.** Whenever a system like a cellular automaton (or PDE) has a global conserved quantity there must always be a local conservation law which expresses the fact that every point in the system the total flux of the conserved quantity into a particular region must equal the rate of increase of the quantity inside it. (If the conserved quantity is thought of like charge, the flux is then current.) In any 1D $k = 2$, $r = 1$ cellular automaton, it follows from the basic structure of the rule that one can tell what the difference in values of a particular cell on two successive steps will be just by looking at the cell and its immediate neighbor on each side. But if the number of black cells is conserved, then one can compute this difference instead by defining a suitable flux, and subtracting its values on the left and right of the cell. What the flux should be depends on the rule. For rule 184, it can be taken to be 1 for each ■□ block, and to be 0 otherwise. For rule 170, it is 1 for both □□ and ■□. For rule 150, it is 1 for □□ and ■■, with all computations done modulo 2. In general, if the global conserved quantity involves blocks of size b, the flux can be computed by looking at blocks of size $b + 2r - 1$. What the values for these blocks should be can be found by solving a system of linear equations; that a solution must exist can be seen by looking at the de Bruijn network (see page 941), with nodes labelled by size $b + 2r - 1$ blocks, and connections by value differences between size b blocks at the center of the possible size $b + 2r$ blocks. (Note that the same basic kind of setup works in any number of dimensions.)

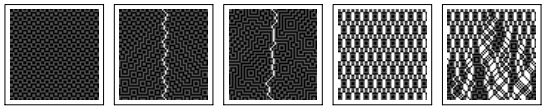
▪ **Block cellular automata.** With a rule of the form *{{1, 1}→{1, 1}, {1, 0}→{1, 0}, {0, 1}→{0, 0}, {0, 0}→{0, 1}}* the evolution of a block cellular automaton with blocks of size n can be implemented using

```
BCAEvolveList[{n_Integer, rule_}, init_, t_] :=
  FoldList[BCAStep[{n, rule}, #1, #2] &, init, Range[t]] /;
    Mod[Length[init], n] == 0
BCAStep[{n_, rule_}, a_, d_] := RotateRight[
    Flatten[Partition[RotateLeft[a, d], n] /. rule], d]
```

Starting with a single black cell, none of the $k = 2$, $n = 2$ block cellular automata generate anything beyond simple nested patterns. In general, there are $k^{n k^n}$ possible rules for block cellular automata with k colors and blocks of size n. Of these, $k^n!$ are reversible. For $k = 2$, the number of rules that conserve the total number of black cells can be computed from *q = Binomial[n, Range[0, n]]* as *Apply[Times, q^q]*. The number of these rules that are also reversible is *Apply[Times, q!]*. In general, a block cellular automaton is reversible only if its rule simply permutes the k^n possible blocks.

Compressing each block into a single cell, and n steps into one, any block cellular automaton with k colors and block size n can be translated directly into an ordinary cellular automaton with k^n colors and range $r = n/2$.

▪ **Page 461 · Block rules.** These pictures show the behavior of rule (c) starting from some special initial conditions.



The repetition period with a total of n cells can be 3^n steps. With random initial conditions, the period is typically up to about $3^{n/2}$. Starting with a block of q black cells, the period can get close to this. For $n = 20$, $q = 17$, for example, it is 31,300.

Note that even in rule (b) wraparound phenomena can lead to repetition periods that increase rapidly with n (e.g. 4820 for $n = 20$, $q = 15$), but presumably not exponentially.

In rule (d), the repetition periods can typically be larger than in rule (c): e.g. 803,780 for $n = 20$, $q = 13$.

▪ **Page 464 · Limiting procedures.** Several different limiting procedures all appear to yield the same continuum behavior for the cellular automata shown here. In the pictures on this

page a large ensemble of different initial conditions is considered, and the density of each individual cell averaged over this ensemble is computed. In a more direct analogy to actual physical systems, one would consider instead a very large number of cells, then compute the density in a single state of the system by averaging over regions that contain many cells but are nevertheless small compared to the size of the whole system.

■ **PDE approximations.** Cellular automaton (d) in the main text can be viewed as minimal discrete approximations to the diffusion equation. The evolution of densities in the ensemble average is analogous to a traditional finite difference method with a real number at each site. The cellular automaton itself uses in effect a distributed representation of the density.

■ **Diffusion equation.** In an appropriate limit the density distribution for cellular automaton (d) appears to satisfy the usual diffusion equation $\partial_t f[x, t] == c\, \partial_{xx} f[x, t]$ discussed on page 163. The solution to this equation with an impulse initial condition is $Exp[-x^2/t]$, and with a block from $-a$ to a it is $(Erf[(a-x)/\sqrt{t}\,] + Erf[(a+x)/\sqrt{t}\,])/a$.

■ **Derivation of the diffusion equation.** With some appropriate assumptions, it is fairly straightforward to derive the usual diffusion equation from a cellular automaton. Let the density of black cells at position x and time t be $f[x, t]$, where this density can conveniently be computed by averaging over many instances of the system. If we assume that the density varies slowly with position and time, then we can make series expansions such as

$$f[x + dx, t] == f[x, t] + \partial_x f[x, t]\, dx + 1/2\, \partial_{xx} f[x, t]\, dx^2 + \ldots$$

where the coordinates are scaled so that adjacent cells are at positions $x - dx$, x, $x + dx$, etc. If we then assume perfect underlying randomness, the density at a particular position must be given in terms of the densities at neighboring positions on the previous step by

$$f[x, t + dt] == p_1\, f[x - dx, t] + p_2\, f[x, t] + p_3\, f[x + dx, t]$$

Density conservation implies that $p_1 + p_2 + p_3 == 1$, while left-right symmetry implies $p_1 == p_3$. And from this it follows that

$$f[x, t + dt] == c\,(f[x - dx, t] + f[x + dx, t]) + (1 - 2\,c)\, f[x, t]$$

Performing a series expansion then yields

$$f[x, t] + dt\, \partial_t f[x, t] == f[x, t] + c\, dx^2\, \partial_{xx} f[x, t]$$

which in turn gives exactly the usual 1D diffusion equation $\partial_t f[x, t] == \xi\, \partial_{xx} f[x, t]$, where ξ is the diffusion coefficient for the system. I first gave this derivation in 1986, together with extensive generalizations.

■ **Page 464 · Non-standard diffusion.** To get ordinary diffusion behavior of the kind that occurs in gases—and is described by the diffusion equation—it is in effect necessary to have perfect uncorrelated randomness, with no structure that persists too long. But for example in the rule (a) picture on page 463 there is in effect a block of solid that persists in the middle—so that no ordinary diffusion behavior is seen. In rule (c) there is considerable apparent randomness, but it turns out that there are also fluctuations that last too long to yield ordinary diffusion. And thus for example whenever there is a structure containing s identical cells (as on page 462), this typically takes about s^2 steps to decay away. The result is that on page 464 the limiting form of the average behavior does not end up being an ordinary Gaussian.

■ **Conservation of vector quantities.** Conservation of the total number of colored cells is analogous to conservation of a scalar quantity such as energy or particle number. One can also consider conservation of a vector quantity such as momentum which has not only a magnitude but also a direction. Direction makes little sense in 1D, but is meaningful in 2D. The 2D cellular automaton used as a model of an idealized gas on page 446 provides an example of a system that can be viewed as conserving a vector quantity. In the absence of fixed scatterers, the total fluxes of particles in the horizontal and the vertical directions are conserved. But in a sense there is too much conservation in this system, and there is no interaction between horizontal and vertical motions. This can be achieved by having more complicated underlying rules. One possibility is to use a hexagonal rather than square grid, thereby allowing six particle directions rather than four. On such a grid it is possible to randomize microscopic particle motions, but nevertheless conserve overall momenta. This is essentially the model used in my discussion of fluids on page 378.

Ultimate Models for the Universe

■ **History of ultimate models.** From the earliest days of Greek science until well into the 1900s, it seems to have often been believed that an ultimate model of the universe was not far away. In antiquity there were vague ideas about everything being made of elements like fire and water. In the 1700s, following the success of Newtonian mechanics, a common assumption seems to have been that everything (with the possible exception of light) must consist of tiny corpuscles with gravity-like forces between them. In the 1800s the notion of fields—and the ether—began to develop, and in the 1880s it was suggested that atoms might be knotted vortices in the ether (see page 1044). When the electron was discovered in 1897 it was briefly thought that it might be the fundamental constituent of everything. And later it was imagined that perhaps electromagnetic fields could underlie

everything. Then after the introduction of general relativity for the gravitational field in 1915, there were efforts, especially in the 1930s, to introduce extensions that would yield unified field theories of everything (see page 1028). By the 1950s, however, an increasing number of subatomic particles were being found, and most efforts at unification became considerably more modest. In the 1960s the quark model began to explain many of the particles that were seen. Then in the 1970s work in quantum field theory encouraged the use of gauge theories and by the late 1970s the so-called Standard Model had emerged, with the Weinberg-Salam $SU(2) \otimes U(1)$ gauge theory for weak interactions and electromagnetism, and the QCD SU(3) gauge theory for strong interactions. The discoveries of the c quark, τ lepton and b quark were largely unexpected, but by the late 1970s there was widespread enthusiasm for the idea of a single "grand unified" gauge theory, based say on SU(5), that would explain all forces except gravity. By the mid-1980s failure to observe expected proton decay cast doubts on simple versions of such models, and various possibilities based on supersymmetry and groups like SO(10) were considered. Occasional attempts to construct quantum theories of gravity had been made since the 1930s, and in the late 1980s these began to be pursued more vigorously. In the mid-1980s the discovery that string theory could be given various mathematical features that were considered desirable made it emerge as the main hope for an ultimate "theory of everything". But despite all sorts of elegant mathematical work, the theory remains rather distant from observed features of our universe. In some parts of particle physics, it is still sometimes claimed that an ultimate theory is not far away, but outside it generally seems to be assumed that physics is somehow instead an endless frontier—that will continue to yield a stream of surprising and increasingly complex discoveries forever—with no ultimate theory ever being found.

▪ **Theological implications.** Some may view an ultimate model of the universe as "leaving no room for a god", while others may view it as a direct reflection of the existence of a god. In any case, knowing a complete and ultimate model does make it impossible to have miracles or divine interventions that come from outside the laws of the universe—though working out what will happen on the basis of these laws may nevertheless be irreducibly difficult.

▪ **Origins of physical models.** Considering the reputation of physics as an empirical science, it is remarkable how many significant theories were in fact first constructed on largely aesthetic grounds. Notable examples include Maxwell's equations for electromagnetism (1880s), general relativity (1915), the Dirac equation for relativistic electrons (1928), and QCD (early 1970s). This history makes it seem more plausible that one might be able to come up with an ultimate model of physics on largely aesthetic grounds, rather than mainly by working from detailed experimental observations.

▪ **Simplicity in scientific models.** To curtail absurdly complicated early scientific models Occam's razor principle that "entities should not be multiplied beyond necessity" was introduced in the 1300s. This principle has worked well in physics, where it has often proven to be the case, for example, that out of all possible terms in an equation the only ones that actually occur are the very simplest. But in a field like biology, the principle has usually been regarded as much less successful. For many complicated features are seen in biological organisms, and when there have been guesses of simple explanations for them, these have often turned out to be wrong. Much of what is seen is probably a reflection of complicated details of the history of biological evolution. But particularly after the discoveries in this book it seems likely that at least some of what appears complicated may actually be produced by very simple underlying programs—which perhaps occur because they were the first to be tried, or are the most efficient or robust. Outside of natural science, Occam's principle can sometimes be useful—typically because simplicity is a good assumption in some aspect of human behavior or motivation. In looking at well-developed technological systems or human organizations simplicity is also quite often a reasonable assumption—since over the course of time parts that are complicated or difficult to understand will tend to have been optimized away.

▪ **Numerology.** Ever since the Pythagoreans many attempts to find truly ultimate models of the universe have ended up centering on derivations of numbers that are somehow thought to be characteristic of the universe. In the past century, the emphasis has been on physical constants such as the fine structure constant $\alpha \simeq 1/137.0359896$, and usually the idea is that such constants arise directly from counting objects of some specified type using traditional discrete mathematics. A notable effort along these lines was made by Arthur Eddington in the mid-1930s, and certainly over the past twenty or so years I have received a steady stream of mail presenting such notions with varying degrees of obscurity and mysticism. But while I believe that every feature of our universe does indeed come from an ultimate discrete model, I would be very surprised if the values of constants which happen to be easy for us to measure in the end turn out to be given by simple traditional mathematical formulas.

▪ **Emergence of simple laws.** In statistical physics it is seen that universal and fairly simple overall laws often emerge

even in systems whose underlying molecular or other structure can be quite complicated. The basic origin of this phenomenon is the averaging effect of randomness discussed in Chapter 7 (technically, it is the survival only of leading operators at renormalization group fixed points). The same phenomenon is also seen in quantum field theory, where it is essentially a consequence of the averaging effect of quantum fluctuations, which have a direct mathematical analog to statistical physics.

▪ **Apparent simplicity.** Given any rules it is always possible to develop a form of description in which these rules will be considered simple. But what is interesting to ask is whether the underlying rules of the universe will seem simple—or special, say in their elegance or symmetry—with respect to forms of description that we as humans currently use.

▪ **Mechanistic models.** Until quite recently, it was generally assumed that if one were able to get at the microscopic constituents of the universe they would look essentially like small-scale analogs of objects familiar from everyday life. And so, for example, the various models of atoms from the end of the 1800s and beginning of the 1900s were all based on familiar mechanical systems. But with the rise of quantum mechanics it came to be believed throughout mainstream physics that any true fundamental model must be abstract and mathematical—and never ultimately amenable to any kind of direct mechanistic description. Occasionally there have been mechanistic descriptions used—as in the parton and bag models, and various continuum models of high-energy collisions—but they have typically been viewed only as convenient rough approximations. (Feynman diagrams may also seem superficially mechanistic, but are really just representations of quite abstract mathematical formulas.) And indeed since at least the 1960s mechanistic models have tended to carry the stigma of uninformed amateur science.

With the rise of computers there began to be occasional discussion—though largely outside of mainstream science—that the universe might have a mechanism related to computers. Since the 1950s science fiction has sometimes featured the idea that the universe or some part of it—such as the Earth—could be an intentionally created computer, or that our perception of the universe could be based on a computer simulation. Starting in the 1950s a few computer scientists considered the idea that the universe might have components like a computer. Konrad Zuse suggested that it could be a continuous cellular automaton; Edward Fredkin an ordinary cellular automaton (compare page 1027). And over the past few decades—normally in the context of amateur science—there have been a steady stream of systems like cellular automata constructed to have elements reminiscent of observed particles or forces. From the point of view of mainstream physics, such models have usually seemed quite naive. And from what I say in the main text, no such literal mechanistic model can ever in the end realistically be expected to work. For if an ultimate model is going to be simple, then in a sense it cannot have room for all sorts of elements that are immediately recognizable in terms of everyday known physics. And instead I believe that what must happen relies on the phenomena discovered in this book—and involves the emergence of complex properties without any obvious underlying mechanistic set up. (Compare page 860.)

▪ **The Anthropic Principle.** It is sometimes argued that the reason our universe has the characteristics it does is because otherwise an intelligence such as us could not have arisen to observe it. But to apply such an argument one must among other things assume that we can imagine all the ways in which intelligence could conceivably operate. Yet as we have seen in this book it is possible for highly complex behavior—ultimately not dissimilar to intelligence—to arise from simple programs in ways that we never came even close to imagining. And indeed, as we discuss in Chapter 12, it seems likely that above a fairly low threshold the vast majority of underlying rules can in fact in some way or another support arbitrarily complex computations—potentially allowing something one might call intelligence in a vast range of very different universes. (See page 822.)

▪ **Physics versus mathematics.** Theoretical physics can be viewed as taking physical input in the form of models and then using mathematics to work out the consequences. If I am correct that there is a simple underlying program for the universe, then this means that theoretical physics must at some level have only a very small amount of true physical input—and the rest must in a sense all just be mathematics.

▪ **Initial conditions.** To find the behavior of the universe one potentially needs to know not only its rule but also its initial conditions. Like the rule, I suspect that the initial conditions will turn out to be simple. And ultimately there should be traces of such simplicity in, say, the distribution of galaxies or the cosmic microwave background. But ideas like those on page 1055—as well as inflation—tend to suggest that we currently see only a tiny fraction of the whole universe, making it very difficult for example to recognize overall geometrical regularities. And it could also be that even though there might ultimately have been simple initial conditions, the current phase of our universe might be the result of some sequence of previous phases, and so effectively have much more complicated initial conditions. (Proposals discussed in quantum cosmology since the 1980s

that for example just involve requiring the universe to satisfy final but not initial boundary condition constraints do not fit well into my kinds of models.)

▪ **Consequences of an ultimate model.** Even if one knows an ultimate model for the universe, there will inevitably be irreducible difficulty in working out all its consequences. Indeed, questions like "does there exist a way to transmit information faster than light?" may boil down to issues analogous to whether it is possible to construct a configuration that has a certain property in, say, the rule 110 cellular automaton. And while some such questions may be answered by fairly straightforward computational or mathematical means, there will be no upper bound on the amount of effort that it could take to answer any particular question.

▪ **Meaning of the universe.** If the whole history of our universe can be obtained by following definite simple rules, then at some level this history has the same kind of character as a construct such as the digit sequence of π. And what this suggests is that it makes no more or less sense to talk about the meaning of phenomena in our universe as it does to talk about the meaning of phenomena in the digit sequence of π.

The Nature of Space

▪ **History of discrete space.** The idea that matter might be made up of discrete particles existed in antiquity (see page 876), and occasionally the notion was discussed that space might also be discrete—and that this might for example be a way of avoiding issues like Zeno's paradox. In 1644 René Descartes proposed that space might initially consist of an array of identical tiny discrete spheres, with motion then occurring through chains of these spheres going around in vortices—albeit with pieces being abraded off. But with the rise of calculus in the 1700s all serious fundamental models in physics began to assume continuous space. In discussing the notion of curved space, Bernhard Riemann remarked in 1854 that it would be easier to give a general mathematical definition of distance if space were discrete. But since physical theories seemed to require continuous space, the necessary new mathematics was developed and almost universally used—though for example in 1887 William Thomson (Kelvin) did consider a discrete foam-like model for the ether (compare page 988). Starting in 1930, difficulties with infinities in quantum field theory again led to a series of proposals that spacetime might be discrete. And indeed by the late 1930s this notion was fairly widely discussed as a possible inevitable feature of quantum mechanics. But there were problems with relativistic invariance, and after ideas of renormalization developed in the 1940s, discrete space seemed unnecessary, and has been out of favor ever since. Some non-standard versions of quantum field theory involving discrete space did however continue to be investigated into the 1960s, and by then a few isolated other initiatives had arisen that involved discrete space. The idea that space might be defined by some sort of causal network of discrete elementary quantum events arose in various forms in work by Carl von Weizsäcker (ur-theory), John Wheeler (pregeometry), David Finkelstein (spacetime code), David Bohm (topochronology) and Roger Penrose (spin networks; see page 1055). General arguments for discrete space were also sometimes made—notably by Edward Fredkin, Marvin Minsky and to some extent Richard Feynman—on the basis of analogies to computers and in particular the idea that a given region of space should contain only a finite amount of information. In the 1980s approximation schemes such as lattice gauge theory and later Regge calculus (see page 1054) that take space to be discrete became popular, and it was occasionally suggested that versions of these could be exact models. There have been a variety of continuing initiatives that involve discrete space, with names like combinatorial physics—but most have used essentially mechanistic models (see page 1026), and none have achieved significant mainstream acceptance. Work on quantum gravity in the late 1980s and 1990s led to renewed interest in the microscopic features of spacetime (see page 1054). Models that involve discreteness have been proposed—most often based on spin networks—but there is usually still some form of continuous averaging present, leading for example to suggestions very different from mine that perhaps this could lead to the traditional continuum description through some analog of the wave-particle duality of elementary quantum mechanics. I myself became interested in the idea of completely discrete space in the mid-1970s, but I could not find a plausible framework for it until I started thinking about networks in the mid-1980s.

▪ **Planck length.** Even in existing particle physics it is generally assumed that the traditional simple continuum description of space must break down at least below about the Planck length $Sqrt[\hbar G/c^3] \approx 2 \times 10^{-35}$ meters—since at this scale dimensional analysis suggests that quantum effects should be comparable in magnitude to gravitational ones.

▪ **Page 472 · Symmetry.** A system like a cellular automaton that consists of a large number of identical cells must in effect be arranged like a crystal, and therefore must exhibit one of the limited number of possible crystal symmetries in any particular dimension, as discussed on page 929. And even a

generalized cellular automaton constructed say on a Penrose tiling still turns out to have a discrete spatial symmetry.

■ **Page 474 · Space and its contents.** A number of somewhat different ideas about space were discussed in antiquity. Around 375 BC Plato vaguely suggested that the universe might consist of large numbers of abstract polyhedra. A little later Aristotle proposed that space is set up so as to provide a definite place for everything—and in effect to force it there. But in geometry as developed by Euclid there was at least a mathematical notion of space as a kind of uniform background. And by sometime after 300 BC the Epicureans developed the idea of atoms of matter existing in a mostly featureless void of space. In the Middle Ages there was discussion about how the non-material character of God might fit in with ideas about space. In the early 1600s the concept of inertia developed by Galileo implied that space must have a certain fundamental uniformity. And with the formulation of mechanics by Isaac Newton in 1687 space became increasingly viewed as something purely abstract, quite different in character from material objects which exist in it. Philosophers had meanwhile discussed matter—as opposed to mind—being something characterized by having spatial extent. And for example in 1643 Thomas Hobbes suggested that the whole universe might be made of the same continuous stuff, with different densities of it corresponding to different materials, and geometry being just an abstract idealization of its properties. But in the late 1600s Gottfried Leibniz suggested instead that everything might consist of discrete monads, with space emerging from the pattern of relative distances between them. Yet with the success of Newtonian mechanics such ideas had by the late 1700s been largely forgotten—leading space almost always to be viewed just in simple abstract geometrical terms. The development of non-Euclidean geometry in the mid-1800s nevertheless suggested that even at the level of geometry space could in principle have a complicated structure. But in physics it was still assumed that space itself must have a standard fixed Euclidean form—and that everything in the universe must just exist in this space. By the late 1800s, however, it was widely believed that in addition to ordinary material objects, there must throughout space be a fluid-like ether with certain mechanical and electromagnetic properties. And in the 1860s it was even suggested that perhaps atoms might just correspond to knots in this ether (see page 1044). But this idea soon fell out of favor, and when relativity theory was introduced in 1905 it emphasized relations between material objects and in effect always treated space as just some kind of abstract background, with no real structure of its own. But in 1915 general relativity introduced the idea that space could actually have a varying non-Euclidean geometry—and that this could represent gravity. Yet it was still assumed that matter was something different—that for example had to be represented separately by explicit terms in the Einstein equations. There were nevertheless immediate thoughts that perhaps at least electromagnetism could be like gravity and just arise from features of space. And in 1918 Hermann Weyl suggested that this could happen through local variations of scale or "gauge" in space, while in the 1920s Theodor Kaluza and Oskar Klein suggested that it could be associated with a fifth spacetime dimension of invisibly small extent. And from the 1920s to the 1950s Albert Einstein increasingly considered the possibility that there might be a unified field theory in which all matter would somehow be associated with the geometry of space. His main specific idea was to allow the metric of spacetime to be non-symmetric (see page 1052) and perhaps complex—with its additional components yielding electromagnetism. And he then tried to construct nonlinear field equations that would show no singularities, but would have solutions (perhaps analogous to the geons discussed on page 1054) that would exhibit various discrete features corresponding to particles—and perhaps quantum effects. But with the development of quantum field theory in the 1920s and 1930s most of physics again treated space as fixed and featureless—though now filled with various types of fields, whose excitations were set up to correspond to observed types of particles. Gravity has never fit very well into this framework. But it has always still been expected that in an ultimate quantum theory of gravity space will have to have a structure that is somehow like a quantum field. But when quantum gravity began to be investigated in earnest in the 1980s (see page 1054) most efforts concentrated on the already difficult problem of pure gravity—and did not consider how matter might enter. In the development of ordinary quantum field theories, supergravity theories studied in the 1980s did nominally support particles identified with gravitons, but were still formulated on a fixed background spacetime. And when string theory became popular in the 1980s the idea was again to have strings propagating in a background spacetime—though it turned out that for consistency this spacetime had to satisfy the Einstein equations. Consistency also typically required the basic spacetime to be 10-dimensional—with the reduction to observed 4D spacetime normally assumed to occur through restriction of the other dimensions to some kind of so-called Calabi-Yau manifold of small extent, associated excitations with various particles through an analog of the Kaluza-Klein mechanism. It has always been hoped that this kind of seemingly arbitrary setup would somehow automatically

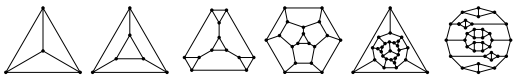
emerge from the underlying theory. And in the late 1990s there seemed to be some signs of this when dualities were discovered in various generalized string theories—notably for example between quantum particle excitations and gravitational black hole configurations. So while it remains impossible to work out all the consequences of string theories, it is conceivable that among the representations of such theories there might be ones in which matter can be viewed as just being associated with features of space.

Space as a Network

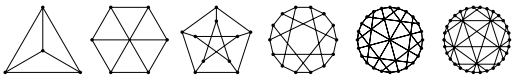
▪ **Page 476 · Trivalent networks.** With n nodes and 3 connections at each node a network must always have an even number of nodes, and a total of $3n/2$ connections. Of all possible such networks, most large ones end up being connected. The number of distinct such networks for even n from 2 to 10 is $\{2, 5, 17, 71, 388\}$. If no self connections are allowed then these numbers become $\{1, 2, 6, 20, 91\}$, while if neither self nor multiple connections are allowed (yielding what are often referred to as cubic or 3-regular graphs), the numbers become $\{0, 1, 2, 5, 19, 85, 509, 4060, 41301, 510489\}$, or asymptotically $(6n)!/((3n)!(2n)!288^n e^2)$. (For symmetric graphs see page 1032.) If one requires the networks to be planar the numbers are $\{0, 1, 1, 3, 9, 32, 133, 681, 3893, 24809, 169206\}$. If one looks at subnetworks with dangling connections, the number of these up to size 10 is $\{2, 5, 7, 22, 43, 141, 373, 1270, 4053, 14671\}$, or $\{1, 1, 2, 6, 10, 29, 64, 194, 531, 1733\}$ if no self or multiple connections are allowed (see also page 1039).

▪ **Properties of networks.** Over the past century or so a variety of global properties of networks have been studied. Typical ones include:

- Edge connectivity: the minimum number of connections that must be removed to make the network disconnected.
- Diameter: the maximum distance between any two nodes in the network. The pictures below show the largest planar trivalent networks with diameters 1, 2 and 3, and the largest known ones with diameters 4, 5 and 6.



- Circumference: the length of the longest cycle in the network. Although difficult to determine in particular cases, many networks allow so-called Hamiltonian cycles that include every node. (Up to 8 nodes, all 8 trivalent networks have this property; up to 10 nodes 25 of 27 do.)
- Girth: the length of the shortest cycle in the network. The pictures below show the smallest trivalent networks with girths 3 through 8 (so-called cages). Girth can be relevant in seeing whether a particular cluster can ever occur in network.



- Chromatic number: the minimum of colors that can be assigned to nodes so that no adjacent nodes end up the same color. It follows from the Four-Color Theorem that the maximum for planar networks is 4. It turns out that for all trivalent networks the maximum is also 4, and is almost always 3.

▪ **Regular polytopes.** In 3D, of the five regular polyhedra, only the tetrahedron, cube and dodecahedron have three edges meeting at each vertex, corresponding to a trivalent network. (Of the 13 additional Archimedean solids, 7 yield trivalent networks.) In 4D the six regular polytopes have 4, 4, 6, 8, 4 and 12 edges meeting at each vertex, and in higher dimensions the simplex ($d+1$ vertices) and hypercube (2^d vertices) have d edges meeting at each vertex, while the co-cube ($2d$ vertices) has $2(d-1)$. (See also symmetric graphs on page 1032, and page 929.)

▪ **Page 476 · Generalizations.** Almost any kind of generalized network can be emulated by a trivalent network just by introducing more nodes. As indicated in the main text, networks with more than three connections at each node can be emulated by combining nodes into groups, and looking only at the connections between groups. Networks with colored nodes can be emulated by representing each color of node by a fixed group of nodes. Going beyond ordinary networks, one can consider hypernetworks in which connections join not just pairs of nodes, but larger numbers of nodes. Such hypernetworks are specified by adjacency tensors rather than adjacency matrices. But it is possible to emulate any hypernetwork by having each generalized connection correspond to a group of connections in an ordinary trivalent network.

▪ **Maintaining simple rules.** An important reason for considering models based solely on trivalent networks is that they allow simpler evolution rules to be maintained (see page 508). If nodes can have more than three connections, then they will often be able to evolve to have any number of connections—in which case one must give what is in effect an infinite set of rules to specify what to do for each number of connections.

■ **Page 477 · 3D network.** The 3D network (c) can be laid out in space using *Array[x[8{##}] &, {n, n, n}]* where

```
x[m:{_, _, _}] := {x1[m], x1[m + 4],
    x2[m + {4, 2, 0}], x2[m + {0, 6, 4}]}
x1[m:{_, _, _}] := Line[Map[# + m &, {{1, 0, 0}, {1, 1, 1},
    {0, 2, 1}, {1, 1, 1}, {3, 1, 3}, {3, 0, 4}, {3, 1, 3}, {4, 2, 3}}]]
x2[{i_, j_, k_}] :=
  x1[{-i - 4, -j - 2, k}] /. {a_, b_, c_} → {-a, -b, c}
```

The resulting structure is a cubic array of blocks with each block containing 8 nodes. The shortest cycle that returns to a particular node turns out to involve 10 edges. The structure does not correspond to the way that chemical bonds are arranged in any common crystalline materials, probably because it would be likely to be mechanically unstable.

■ **Continuum limits.** For all everyday purposes a region in a network with enough nodes and an appropriate pattern of connections can act just like ordinary continuous space. But at a formal mathematical level this can happen rigorously only in an infinite limit. And in general, there is no reason to expect that all properties of the system (notably for example the existence of particles) will be preserved by taking such a limit. But in understanding the structure of space and comparing to ordinary continuous space it is convenient to imagine taking such a limit. Inevitably there are several scales involved, and one can only expect continuum behavior if one looks at scales intermediate between individual connections in the underlying network and the overall size of the whole network. Yet as I will discuss on pages 534 and 1050 even at such scales it is far from straightforward to see how all the various well-studied properties of ordinary continuous space (as embodied for example in the theory of manifolds) can emerge from discrete underlying networks.

■ **Page 478 · Definitions of distance.** Any measure of distance—whether in ordinary continuous space or elsewhere—takes a pair of points and yields a number. Several properties are normally assumed. First, that if the points are identical the distance is zero, and if they are different, it is a positive number. Second, that the distance between points A and B is the same as between B and A. And third, that the so-called triangle inequality holds, so that the distance AC is no greater than the sum of the distances AB and BC. With distance on a network defined as the length of shortest path between nodes one immediately gets all three of these properties. And even though all distances defined this way will be integers, they still make any network formally correspond in mathematical terms to a metric space (or strictly a path metric space). If the connections on the underlying network are one-way (as in causal networks) then one no longer necessarily gets the second property, and when a continuum limit exists it can correspond to a (perhaps discontinuous) section through a fiber bundle rather than to a manifold. Note that as discussed on page 536 physical measures of distance will always end up being based not just on single paths in a network, but on the propagation of something like a particle, which typically in effect requires the presence of many paths. (See page 1048.)

■ **Page 478 · Definitions of dimension.** The most obvious way to define the dimension of a space is somehow to ask how many parameters—or coordinates—are needed to specify a point in it. But starting in the 1870s the discovery of constructs like space-filling curves (see page 1127) led to investigation of other definitions. And indeed there is some reason to believe that around 1884 Georg Cantor may have tried developing a definition based on essentially the idea that I use here of looking at growth rates of volumes of spheres (balls). But for standard continuous spaces this definition is hard to make robust—since unlike in discrete networks where one can define volume just by counting nodes, defining volume in a continuous space requires assigning a potentially arbitrary density function. And as a result, in the late 1800s and early 1900s other definitions of dimension were developed. What emerged as most popular is topological dimension, in which one fills space with overlapping balls, and asks what the minimum number that ever have to overlap at any point will be. Also considered was so-called Hausdorff dimension, which became popular in connection with fractals in the 1980s (see page 933), and which can have non-integer values. But for discrete networks the standard definitions for both topological and Hausdorff dimension give the trivial result 0. One can get more meaningful results by thinking about continuum limits, but the definition of dimension that I give in the main text seems much more straightforward. Even here, there are however some subtleties. For example, to find a definite volume growth rate one does still need to take some kind of limit—and one needs to avoid sampling too many or too few nodes in the network. And just as with fractal dimensions discussed on page 933 there are issues about whether a definite power law for the growth rate will emerge, and how one should average over results for different parts of the network. There are some alternative approaches to defining dimension in which some of these issues at least become less explicit. For example, one can imagine not just forming a ball on the network, but instead growing something like a cellular automaton, and seeing how big a pattern it produces after some number of steps. And similarly, one can for example look at the statistics of random walks on the network. A slightly different but still related approach is to study the

density of eigenvalues of the Laplace operator—which can also be thought of as measuring the number of solutions to equations giving linear constraints on numbers assigned to connected nodes. More sophisticated versions of this involve looking at invariants studied in topological field theory. And there are potentially also definitions based for example on considering geodesics and seeing how many linearly independent directions can be defined with them. (Note that given explicit coordinates, one can check whether one is in d or more dimensions by asking for all possible points

```
Det[Table[(x[i] - x[j]) . (x[i] - x[j]), {i, d + 3}, {j, d + 3}]] == 0
```

and this should also work for sufficiently separated points on networks. Still another related approach is to consider coloring the edges of a network: if there are $d + 1$ possible colors, all of which appear at every node, then it follows that d coordinates can consistently be assigned to each node.)

■ **Page 478 · Counting of nodes.** The number of nodes reached by going out to network distance r (with $r > 1$) from any node in the networks on page 477 is (a) $4r - 4$, (b) $3r^2/2 - 3r/2 + 1$, and (c)

```
First[Select[4 r^3/9 + 2 r^2/3 +
    {2, 5/3, 5/3} r - {10/9, 1, -4/9}, IntegerQ]]
```

In any trivalent network, the quantity $f[r]$ obtained by adding up the numbers of nodes reached by going distance r from each node must satisfy $f[0] = n$ and $f[1] = 3n$, where n is the total number of nodes in the network. In addition, the limit of $f[r]$ for large r must be n^2. The values of $f[r]$ for all other r will depend on the pattern of connections in the network.

■ **Page 479 · Cycle lengths.** The lengths of the shortest cycles (girths) of the networks on page 479 are (a) 3, (b) 5, (c) 4, (d) 4, (e) 3, (f) 5, (g) 6, (h) 10, (i) ∞, (j) 3. Note that rules of the kind discussed on page 508 which involve replacing clusters of nodes can only apply when cycles in the cluster match those in the network.

■ **Page 479 · Volumes of spheres.** See page 1050.

■ **Page 480 · Implementation.** Networks are conveniently represented by assigning a number to each node, then having lists of rules which specify what nodes the connection from a particular node go to. The tetrahedron network from page 476 is for example given in this representation by

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

The list of nodes reached by following up to n connections from node i are then given by

```
NodeLists[g_, i_, n_] :=
  NestList[Union[Flatten[# /. g]] &, {i}, n]
```

The network distance corresponding to the length of the shortest path between two nodes is given by

```
Distance[g_, {i_, j_}] := Length[NestWhileList[
    Union[Flatten[# /. g]] &, {i}, ! MemberQ[#, j] &]] - 1
```

■ **Finding layouts.** One way to lay out a network g so that network distances in it come as close as possible to ordinary distances in d-dimensional space, is just to search for values of the $x[i, k]$ which minimize a quantity such as

```
With[{n = Length[g]}, Apply[Plus,
  Flatten[(Table[Distance[g, {i, j}], {i, n}, {j, n}]^2 - Table[
      Sum[(x[i, k] - x[j, k])^2, {k, d}], {i, n}, {j, n}])^2]]]
```

using for example *FindMinimum* starting say with $x[1, \_] \to 0$ and all the other $x[\_, \_] \to$ *Random*[]. Rarely is there a unique minimum that can be found, but the approach nevertheless seems to work fairly well whenever a good layout exists in a particular number of dimensions. One can imagine weighting different network distances differently, but usually I have found that equal weightings work best. If one ignores all constraints beyond network distance 1, then one is in effect just trying to build the network out of identical rigid rods. It turns out that this is almost always possible even in 2D (though not in 1D); the only exception is the tetrahedron network. And in fact very few trivalent structures are rigid, in the sense the angles between rods are uniquely determined. (In 3D, for example, this is true only for the tetrahedron.)

■ **Hamming distances.** In the so-called loop switching method of routing messages in communications systems one lays out a network on an m-dimensional Boolean hypercube so that the distance on the hypercube (equal to Hamming distance) agrees with distance in the network. It is known that to achieve this exactly, m must be at the least the number of either positive or negative eigenvalues of the distance matrix for the network, and can need to be as much as $n - 1$, where n is the total number of nodes.

■ **Continuous mathematics.** Even though networks are discrete, it is conceivable that network-based models can also be formulated in terms of continuous mathematics, with a network-like structure emerging for example from the pattern of singularities or topology of continuous surfaces or functions.

The Relationship of Space and Time

■ **History.** The idea of representing time graphically like space has a long history—and was used for example by Nicholas Oresme in the mid-1300s. In the 1700s and 1800s the idea of position and time as just two coordinates was widespread in mathematical physics—and this then led to notions like "travelling in time" in H. G. Wells's 1895 *The Time Machine*. The mathematical framework developed for relativity theory in the early 1900s (see page 1042) treated space and time very

symmetrically, leading popular accounts of the theory to emphasize a kind of fundamental equivalence between them and to try to make this seem inevitable through rather confusing thought experiments on such topics as idealized trains travelling near the speed of light.

In the context of traditional mathematical equations there has never been much reason to consider the possibility that space and time might be fundamentally different. For typically space and time are both just represented by abstract symbolic variables, and the formal process of solving equations as a function of position in space and as a function of time is essentially identical. But as soon as one tries to construct more explicit models of space and time one is immediately led to consider the possibility that they may be quite different.

▪ **Page 482 · Discreteness in time.** In present-day physics, time, like space, is always assumed to be perfectly continuous. But experiments—the most direct of which are based on looking for quantization in the measured decay times of very short-lived particles—have only demonstrated continuity on scales longer than about 10^{-26} seconds, and there is nothing to say that on shorter scales time is not in fact discrete. (The possibility of a discrete quantum of time was briefly discussed in the 1920s when quantum mechanics was first being developed.)

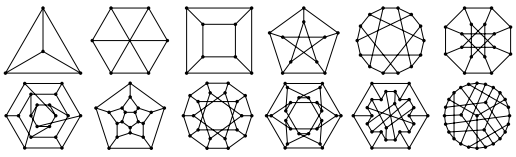
▪ **Page 483 · Network constraint systems.** Cases (a), (f) and (p) allow all networks that do not contain respectively cycles of length 1 (self-loops), cycles of length 3 or less, and cycles of length 5 or less. In cases where an infinite sequence of networks is allowed, there are typically particular subnetworks that can occur any number of times, making the sizes of allowed networks form arithmetic progressions. In cases (m), (n) and (o) respectively triangle, pentagon and square subnetworks can be repeated.

The main text excludes templates that have no dangling connections, and are thus themselves already complete networks. There are 5 such templates involving nodes out to distance one, but of these only 3 correspond to networks that satisfy the constraint that around each node the network has the same form as the template. Among templates involving nodes out to distance two there are 106 that have no dangling connections, and of these only 8 satisfy the constraints.

The main text considers only constraints based on a single template. One can also allow each node to have a neighborhood that corresponds to any of a set of templates. For templates involving nodes out to distance one, there are 13 minimal sets in the sense of page 941, of which only 6 contain just one template, 6 contain two and 1 contains three.

If one does allow dangling connections to be joined within a single template, the results are similar to those discussed so far. There are 52 possible templates involving nodes out to distance two, of which 12 allow complete networks to be formed, none forced to be larger than 12 nodes. There are 46 minimal sets, with the largest containing 4 templates, but none forcing a network larger than 16 nodes.

▪ **Symmetric graphs.** The constraints in a network constraint system require that the structure around each node agrees with a template that contains some number of nodes. A symmetric graph satisfies the same type of constraint, but with the template being the whole network. The pictures below show the smallest few symmetric graphs with 3 connections at each node (with up to 100 nodes there are still only 37 such graphs; compare page 1029).



▪ **Cayley graphs.** As discussed on page 938, the structure of a group can be represented by a Cayley graph where nodes correspond to elements in the group, and connections specify results of multiplying by generators. The transitivity of group multiplication implies that Cayley graphs always have the property of being symmetric (see above). The number of connections at each node is fixed, and given by the number of distinct generators and inverses. In cases such as the tetrahedral group A_4 there are 3 connections at each node. The relations among the generators of a group can be thought of as constraints defining the Cayley graph. As mentioned on page 938, there are finite groups that have simple relations but at least very large Cayley graphs. For infinite groups, it is known (see page 938) that in most cases Cayley graphs are locally like trees, and so do not have finite dimension. It appears that only when the group is nilpotent (so that certain combinations of elements commute much as they do on a lattice) is there polynomial growth in the Cayley graph and thus finite dimension.

▪ **Page 485 · Spacetime symmetric rules.** With $k = 2$ and the neighborhoods shown here, only the additive rules 90R, 105R, 150R and 165R are space-time symmetric. For larger k and larger neighborhoods, there presumably begin to be non-additive rules with this property.

Time and Causal Networks

▪ **Causal networks.** The idea of using networks to represent interdependencies of events seems to have developed with the systematization of manufacturing in the early 1900s—

notably in the work of Frank and Lillian Gilbreth—and has been popular since at least the 1940s. Early applications included switching circuits, logistics planning, decision analysis and general flowcharting. In the last few decades causal networks have been widely used in system specification methods such as Petri nets, as well as in schemes for medical and other diagnosis. Since at least the 1960s, causal networks have also been discussed as representations of connections between events in spacetime, particularly in quantum mechanics (see page 1027).

Causal networks like mine that are ultimately associated with some evolution or flow of activity always have certain properties. In particular, they can never contain loops, and thus correspond to directed acyclic graphs. And from this it follows for example that even the most circuitous path between two nodes must be of finite length.

Causal networks can also be viewed as Hasse diagrams of partially ordered sets, as discussed on page 1040.

■ **Implementation.** Given a list of successive positions of the active cell, as from *Map[Last, MAEvolveList[rule, init, t]]* (see page 887), the network can be generated using

```
MAToNet[list_] := Module[{u, j, k}, u[_] = ∞; Reverse[
    Table[j = list[[i]]; k = {u[j - 1], u[j], u[j + 1]}; u[j - 1] =
      u[j] = u[j + 1] = i; i → k, {i, Length[list], 1, -1}]]]
```

where nodes not yet found by explicit evolution are indicated by ∞.

■ **Page 488 · Mobile automata.** The special structure of mobile automata of the type used here leads to several special features in the causal networks derived from them. One of these is that every node always has exactly 3 incoming and 3 outgoing connections. Another feature is that there is always a path of doubled connections (associated with the active cell) that visits every node in some order. And in addition, the final network must always be planar—as it is whenever it is derived from the evolution of a local underlying 1D system.

■ **Computational compression.** In the model for time described here, it is noteworthy that in a sense an arbitrary amount of underlying computation can take place between successive moments in perceived time.

■ **Page 496 · 2D mobile automata.** As in 2D random walks, active cells in 2D mobile automata often do not return to positions they have visited before, with the result that no causal connections end up being created.

The Sequencing of Events in the Universe

■ **Implementation.** Sequential substitution systems in which only one replacement is ever done at each step can just be implemented using */.* as described on page 893. Substitution systems in which all replacements are done that are found to fit in a left-to-right scan can be implemented as follows

```
GSSEvolveList[rule_, s_, n_] :=
  NestList[GSSStep[rule, #] &, s, n]
GSSStep[rule_, s_] :=
  g[rule, s, f[StringPosition[s, Map[First, rule]]]]
f[{}] = {}; f[s_] := Fold[If[Last[Last[#1]] ≥ First[#2],
      #1, Append[#1, #2]] &, {First[s]}, Rest[s]]
g[rule_, s_, {}] := s; g[rule_, s_, pos_] := StringReplacePart[
    s, Map[StringTake[s, #] &, pos] /. rule, pos]
```

with rules given as *{"ABA" → "BAAB", "BBBB" → "AA"}*.

■ **Generating causal networks.** If every element generated in the evolution of a generalized substitution system is assigned a unique number, then events can be represented for example by *{4, 5} → {11, 12, 13}*—and from a list of such events a causal network can be built up using

```
With[{u = Map[First, list]}, MapIndexed[Function[
    {e, i}, First[i] → Map[(If[# === {}, ∞, #[[1, 1]]] &)[
      Position[u, #]] &, Last[e]]], list]]
```

■ **The sequential limit.** Even when the order of applying rules does not matter, using the scheme of a sequential substitution system will often give different results. If there is a tree of possible replacements (as in *"A" → "AA"*), then the sequential substitution system in a sense does depth-first recursion in the infinite tree, never returning from the single path it takes. Other schemes are closer to breadth-first recursion.

■ **Page 502 · Rule (b).** The maximum number of steps for which the rule can be applied occurs with initial conditions consisting of a white element followed by n black elements, and in this case the number of steps is $2^n + n$.

■ **String theory.** The sequences of symbols I call strings here have absolutely no direct connection to the continuous deformable 1D objects known as strings in string theory.

■ **String overlaps.** The total numbers of strings with length n and k colors that cannot overlap themselves are given by

```
a[0] = 1; a[n_] := k a[n - 1] - If[EvenQ[n], a[n/2], 0]
```

Up to reversal and interchange of A and B, the first few overlap-free strings with 2 colors are A, AB, AAB, $AAAB$, $AABB$.

The shortest pairs of strings of 2 elements with no self- or mutual overlaps are *{"A", "B"}*, *{"AABB", "AABAB"}*, *{"AABB", "ABABB"}*; there are a total of 13 such pairs with strings up to length 5, and 85 with strings up to length 6.

The shortest non-overlapping triple of strings is *{"AAABB", "ABABB", "ABAABB"}* and its variants. There are a total of 36 such triples with no string having length more than 6.

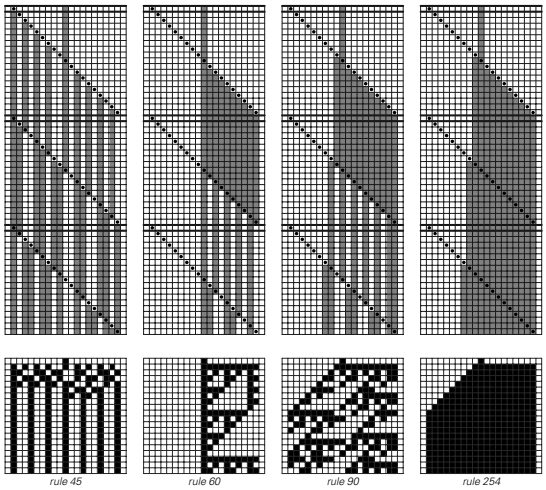
■ **Simulating mobile automata.** Given a mobile automaton like the one from page 73 with rules in the form used on page

887—and behavior of any complexity—the following will yield a causal-invariant substitution system that emulates it:

```
Map[StringJoin, Map[{"AAABB", "ABABB", "ABAABB"}[[
    # + 1]] &, Map[Insert[#[[1]], 2, 2] →
    Insert[#[[2, 1]], 2, 2 + #[[2, 2]]] &, rule], {2}], {2}]
```

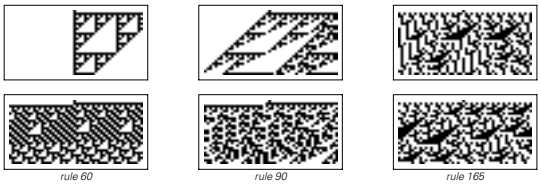
▪ **Sequential cellular automata.** Ordinary cellular automata are set up so that every cell is updated in parallel at each step, based on the colors of neighboring cells on the previous step. But in analogy with generalized substitution systems, one can also consider sequential cellular automata, in which cells are updated sequentially rather than in parallel. The behavior of such systems is usually very different from that of corresponding ordinary cellular automata, mainly because in sequential cellular automata the new color of a particular cell can depend on new rather than old colors of neighboring cells.

The pictures below show the behavior of several sequential cellular automata with $k = 2$, $r = 1$ elementary rules. In the top picture of each pair every individual update is indicated by a black dot. In the bottom picture each line represents one complete step of evolution, including one update of each cell. Note that in this representation, effects can propagate all the way across the system in a single step.

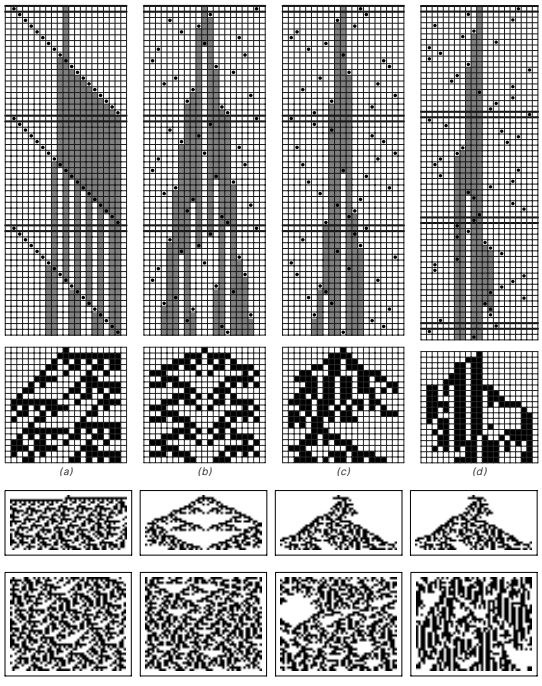


Size dependence. Because effects can propagate all the way across the system in a single step, the overall size, as well as boundary conditions, for the system can be significant after just a few steps, as illustrated in the pictures of rule 60 below.

Additive rules. Among elementary sequential cellular automata, those with additive rules turn out to yield some of the most complex behavior, as illustrated below. The top row shows evolution with the boundary forced to be white; the bottom row shows cyclic boundary conditions. Even though the basic rule is additive, there seems to be no simple traditional mathematical description of the results.



Updating orders. Somewhat different results are typically obtained if one allows different updating orders. For each complete update of a rule 90 sequential cellular automaton, the pictures below show results with (a) left-to-right scan, (b) random ordering of all cells, the same for each pass through the whole system, (c) random ordering of all cells, different for different passes, (d) completely random ordering, in which a particular cell can be updated twice before other cells have even been updated once.



History. Sequential cellular automata have a similar relationship to ordinary cellular automata as implicit updating schemes in finite difference methods have to explicit ones, or as infinite impulse response digital filters have to finite ones. There were several studies of sequential or asynchronous cellular automata done following my work on ordinary cellular automata in the early 1980s.

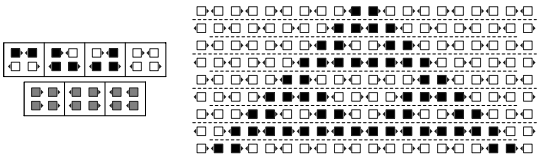
Implementation. The following will update triples of cells in the specified order by using the function *f*:

```
OrderedUpdate[f_, a_, order_] := Fold[ReplacePart[
    #1, f[Take[#1, {#2 - 1, #2 + 1}]], #2] &, a, order]
```

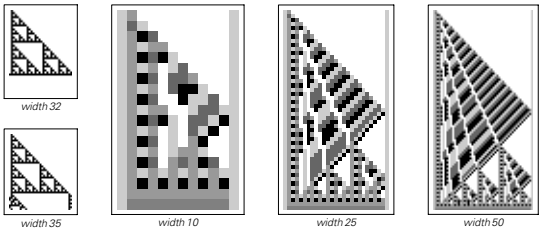
A random ordering of *n* cells corresponds to a random permutation of the form

```
Fold[Insert[#1, #2, Random[Integer, Length[#1]] + 1] &,
  {}, Range[n]]
```

▪ **Intrinsic synchronization in cellular automata.** Taking the rules for an ordinary cellular automaton and applying them sequentially will normally yield very different results. But it turns out that there are variants on cellular automata in which the rules can be applied in any order and the overall behavior obtained—or at least the causal network—is always the same. The picture below shows how this works for a simple block cellular automaton. The basic idea is that to each cell is added an arrow, and any pair of cells is updated only when their arrows point at each other. This in a sense forces cells to wait to be updated until the data they need is ready. Note that the rules can be thought of as replacements such as *"A><B" → "<AB>"* for blocks of length 4 with 4 colors.



▪ **"Firing squad" synchronization.** By choosing appropriate rules it is possible to achieve many forms of synchronization directly within cellular automata. One version posed as a problem by John Myhill in 1957 consists in setting up a rule in which all cells in a region go into a special state after exactly the same number of steps. The problem was first solved in the early 1960s; the solution using 6 colors and a minimal number of steps shown on the right below was found in 1988 by Jacques Mazoyer, who also determined that no similar 4-color solutions exist. Note that this solution in effect constructs a nested pattern of any width (it does this by optionally including or excluding one additional cell at each nesting level, using a mechanism related to the decimation systems of page 909). If one drops the requirement of cells going into a special state, then even the 2-color elementary rule 60 shown on the left can be viewed as solving the problem—but only for widths that are powers of 2.



▪ **Distributed computing.** Many of the basic issues about the progress of time in a universe consisting of many separate elements have analogs in the progress of computations that are distributed across many separate computing elements. In practice, such computations are most often done by requiring explicit synchronization of all elements at appropriate points, and implementing this using a mechanism that is outside of the computation. But more theoretical investigations of formal concurrent systems, temporal logics, dataflow systems, Petri nets and so on have led to ideas about distributed computing that are somewhat closer to the ones I discuss here for the universe. And, as it happens, in the mid-1980s I tried hard, though at the time without much success, to use updating rules for networks as the basis for a new kind of programming language intended for massively parallel computers.

Uniqueness and Branching in Time

▪ **Page 506 · String transformations.** An example of a rule that allows one to go from any string of *A*'s and *B*'s to any other is

```
{"A" → "AA", " AA" → "A", " A" → "B", " B" → "A"}
```

(Compare page 1038.)

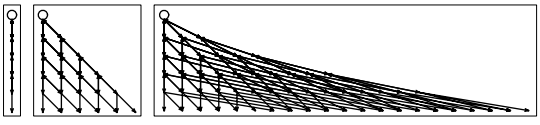
▪ **Parallel universes.** The idea of parallel universes which somehow interact with each other has been much explored in science fiction. And one might think that if the history of each universe corresponds to one path in a multiway system then the convergence of paths might represent interactions between universes. But in fact, much as in the case of time travel, such connections do not represent additional observable effects; they simply imply consistency conditions, in this case between universes whose paths converge.

▪ **Many-worlds models.** The notion of "many-figured time" has been discussed since the 1950s in the context of the many-worlds interpretation of quantum mechanics. There are some similarities to the multiway systems that I consider here. But an important difference is that while in the many-worlds

approach, branchings are associated with possible observation or measurement events, what I suggest here is that they could be an intrinsic feature of even the very lowest-level rules for the universe. (See also page 1063.)

▪ **Spacetime networks from multiway systems.** The main text considers models in which the steps of evolution in a multiway system yield a succession of events in time. An alternative kind of model, somewhat analogous to the ones based on constraints on page 483, is to take the pattern of evolution of a multiway system to define directly a complete spacetime network. Instead of looking separately at strings produced at each step, one instead maintains just a single copy of each distinct string ever produced, and makes that correspond to a node in the network. Each node is then connected to the nodes associated with the strings reached by one application of the multiway rule, as on page 209.

It is fairly straightforward to generate in this way networks of any dimension. For example, starting with n A's the rule $\{"A" \to "AB", "AB" \to "A"\}$ yields a regular n-dimensional grid, as shown below.



If each node in a network is associated with a point in spacetime, then one slightly peculiar feature is that every such point would have an associated string—something like an encoded position coordinate. And it then becomes somewhat difficult to understand why different regions of spacetime seem to behave so similarly—and do not, for example, seem to depend on the details of their coordinates.

▪ **Page 507 · Commuting operations.** If replacements on strings are viewed as mathematical operations, then when the replacements give the same result if applied in any order, the corresponding operations commute.

▪ **Conditions for convergence.** One way to guarantee that there is convergence after one step is to require as in the previous section that blocks to be replaced cannot overlap with themselves or each other. And of the 196 possible rules involving two colors and blocks of length at most three, 112 have this property. But there are also an additional 20 rules which allow some overlap but which nevertheless yield convergence after one step. Examples are $"AAA" \to "A"$ and $"AA" \to "ABA"$. In these rules some of the elements essentially just supply context, but are not affected by the replacement. These elements can then overlap while not affecting the result. Note that unless one excludes the context elements from events, paths in the multiway system will converge, but the causal networks on these paths will be locally slightly different.

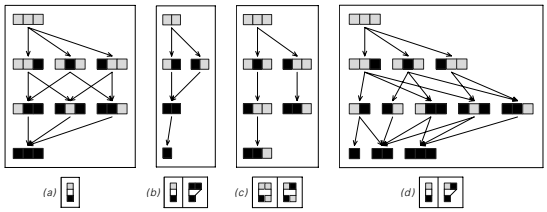
Much as in the previous section, even if paths do not converge for every possible string, it can still be true that paths converge for all strings that are actually generated from a particular initial string.

In general, one can consider convergence after any number of steps, requiring that any two strings which have a common ancestor must at some point also have a common successor. Note that a rule such as $\{"A" \to "B", "A" \to "C", "B" \to "A", "B" \to "D"\}$ exhibits convergence for all paths that have diverged for only one step, but not for all those that have diverged for longer. In general it is formally undecidable whether a particular multiway system will eventually exhibit convergence of all paths.

▪ **Confluence.** As mentioned on page 938, multiway systems have been studied in mathematical logic, typically under names such as rewrite systems, since the early 1900s. The property of path convergence discussed in the main text has been considered since the 1930s, usually under the name of confluence, or sometimes the Church-Rosser property. (Also considered is strong confluence—that paths can always converge in at most one step, and local confluence—that paths can converge after diverging for one step but not necessarily more. Early in its history confluence was most often studied for symbolic systems and lambda calculus rather than ordinary multiway systems.)

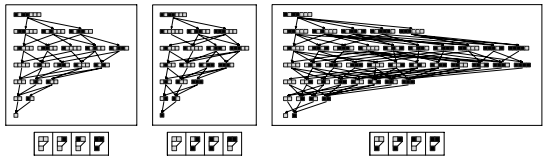
Confluence is important in defining a notion of equivalence for strings. One can say that two strings are equivalent if they can both be transformed to the same string by using the rules of the multiway system. And with such a definition, confluence is what is needed to obtain transitivity for equality, so that $p == q$ and $q == r$ implies $p == r$.

Most often confluence is studied in the context of terminating multiway systems—multiway systems in which eventually strings are produced to which no further replacements apply. If a terminating multiway system has the confluence property, then this implies that regardless of the path taken, a given string will always evolve to a unique string that can be thought of as giving a canonical or normal form for the original string. Examples (a) through (c) below have this property; (d) does not. In example (a), the canonical form is all elements black; in (b) it is a single black element, and in (c) all elements are black, except the last one, which is white if there were any initial white elements. Note that the first example on page 507 has a canonical form consisting of a sorted string.



The process of evaluation in mathematics or in a computer language such as *Mathematica* can be thought of as involving the application of a sequence of replacement rules. Only if these rules have the confluence property will the results always be unique, and independent of the order of rule application.

The evaluation of functions with attribute *Flat* in *Mathematica* provides an example of confluence. If *f* is *Flat*, then in evaluating *f[a, b, c]* one can equally well start with *f[f[a, b], c]* or *f[a, f[b, c]]*. Showing only the arguments to *f*, the pictures below illustrate how the flat functions *Xor* and *And* are confluent, while the non-flat function *Implies* is not.



■ **Completion.** If one has a multiway system that terminates but is not confluent then it turns out often to be possible to make it confluent by adding a finite set of new rules. Given a string p which gets transformed either to q or r by the original rules, one can always imagine adding a new rule $q \to r$ or $r \to q$ that makes the paths from p immediately converge. To do this explicitly for all possible p that can occur would however entail having infinitely many new rules. But as noted by Donald Knuth and Peter Bendix in 1970 it turns out often to be sufficient just iteratively to add new rules only for each so-called critical pair q, r that is obtained from strings p that represent minimal overlaps in the left-hand sides of the rules one has. To decide whether to add $q \to r$ or $r \to q$ in each case one can have some kind of ordering on strings. For the procedure to work this ordering must be such that the strings generated on successive steps in every possible evolution of the multiway system follow the ordering. A number of variations of the basic procedure—using different orderings and with different schemes for dropping redundant rules—have been proposed for systems arising in different kinds of applications. The original Knuth-Bendix procedure was for equations (of the form $a \leftrightarrow b$) had the feature that it could terminate yet not give a confluent multiway system. But in the 1980s so-called unfailing completion algorithms (see page 1158) were developed that—if they terminate—guarantee to give confluent systems. (The question of whether any procedure of this type will terminate in a particular case is nevertheless in general undecidable.)

The basic idea of so-called critical pair completion procedures has arisen several times—notably in the Gröbner basis approach of Bruno Buchberger from 1965 to finding canonical forms for systems of polynomials.

■ **Relationships between types of networks.** Each arrow on each path in a multiway system corresponds to a node in a causal network. Each element in each string in a multiway system corresponds to a connection in a causal network. Each complete string in a multiway system corresponds to a possible slice that goes through all connections across a causal network. Such a slice can be considered in traditional physics terms as a spacelike hypersurface (see page 1041).

Evolution of Networks

■ **Page 509 · Neighbor-independent rules.** Even though the same replacement is performed at each node at each step, the networks produced are not homogeneous. In the first case shown, the picture produced after t steps has $4 \times 3^{t-k-1}$ regions with 3×2^k edges. In the limit $t \to \infty$, the picture has the geometrical form of an Apollonian circle packing (see page 986). The number of nodes at distance up to r from a given node is at most *1 + Sum[c[i] + c[i - 1], {i, n}]* where *c[i_] := 2^DigitCount[i, 2]*. In practice this number fluctuates greatly with r, making pictures like those on page 479 not exhibit smooth profiles. Averaged over all nodes, however, the number of nodes at distance up to r approximates *r^Log[2, 3]*, implying an effective dimension of *Log[2, 3]*. Note that there is no upper limit on the dimension that can be obtained with appropriate neighbor-independent rules.

■ **Implementation.** For many practical purposes the best representation for networks is the one given on page 1031. But in updating networks a particularly straightforward implementation of one scheme can be obtained if one uses instead a more explicit symbolic representation such as

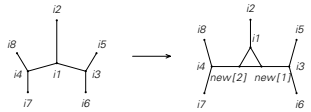
u[1 → v[2, 3, 4], 2 → v[1, 3, 4], 3 → v[1, 2, 4], 4 → v[1, 2, 3]]

This allows one to capture the basic character of networks by

Attributes[u] = {Flat, Orderless}; Attributes[v] = Orderless

Updating rules can then be written in terms of ordinary *Mathematica* patterns. A slight complication is that the patterns have to include all nodes whose connections go to

nodes whose labels are changed by the update. The rule at the top of page 509 must therefore be written out as



and this corresponds to the *Mathematica* rule

```
u[i1_ → v[i2_, i3_, i4_], i3_ → v[i1_, i5_, i6_],
    i4_ → v[i1_, i7_, i8_]] :> u[i1 → v[i2, new[1], new[2]],
    new[1] → v[i1, new[2], i3], new[2] → v[i1, new[1], i4],
    i3 → v[new[1], i5, i6], i4 → v[new[2], i7, i8]]
```

(Strictly there also need to be additional rules to cover where for example nodes 3 and 4 are actually the same.) With rules in this form the network update is simply

```
NetStep[rule_, net_] := Block[{new},
    net /. rule /. new[n_] → n + Apply[Max, Map[First, net]]]
```

Note that just as we discussed for strings on page 1033 the direct use of `/.` here corresponds to a particular scheme for applying the update rule.

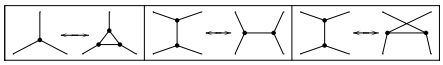
■ **Identifying subnetworks.** The problem of finding where in a network a given subnetwork can occur turns out in general to be computationally difficult. For strings the analogous problem is straightforward, since in a string of length n one can ultimately just try each of the n possible starting points for the substring and see for which of them a match occurs. But for a network with n nodes, a similar procedure would require one to check n^k possible configurations in order to find out where a subnetwork of size k occurs. In practice, however, for fixed subnetworks, one can devise fairly efficient procedures. But the general problem of so-called subgraph isomorphism is formally NP-complete.

■ **Page 509 · Number of replacements.** The total number of distinct replacements that maintain planarity, involve clusters with up to five nodes and have from 3 to 7 dangling connections is $\{16, 8, 125, 24, 246\}$. Not maintaining planarity, the numbers are $\{14, 5, 13, 2, 2\}$. (See page 1039.)

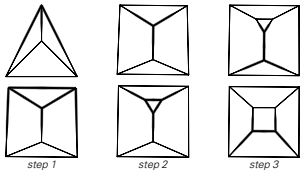
■ **Cycles in networks.** See page 1031.

■ **Planar networks.** One feature of a planar network is that it is always possible to identify definite regions or faces bounded by connections in the network. And from Euler's formula $f + n = e + 2$, it then follows that the average number of edges of each face is always $6(1 - 2/f)$, where f is the total number of faces. Note that with my definition of dimension for networks, the fact that a network is planar does not necessarily mean that it has be two-dimensional—and for example the networks on page 509 are not.

■ **Arbitrary transformations.** By applying the string transformation rules on page 1035 at appropriate locations, it is possible to transform any string of A's and B's to any other. And the analog of this for networks is that by applying the rules shown below at appropriate locations it is possible to transform any network into any other. These rules correspond to the moves invented by James Alexander in 1923 in connection with transforming one knot into another. (Note that the first two rules suffice for all planar networks, and are sometimes called respectively T2 and T1.)



As an example, the pictures below show how a tetrahedron network can be transformed into a cube.



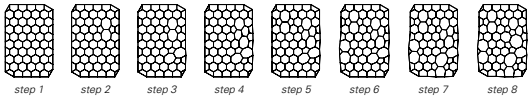
■ **Random networks.** One way to generate the connections for a "completely random" trivalent network with n nodes is just to apply a random permutation:

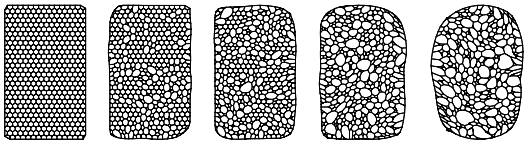
```
RandomNetwork[n_?EvenQ] := Partition[
    Fold[Insert[#1, #2, Random[Integer, Length[#1]] + 1] &,
      {}, Floor[Range[1, n + 2/3, 1/3]]], 2]
```

Networks obtained in this way are usually connected, but will almost always contain self-loops and multiple edges. Properties of random networks are discussed on page 963. A convenient way to get somewhat random planar networks is from 2D Voronoi diagrams of the kind discussed on page 987.

■ **Random replacements.** As indicated in the note above, applying the second rule (T1, shown as (b) on page 511) at an appropriate sequence of positions can transform one planar network into any other with the same number of nodes. The pictures below show what happens if this rule is repeatedly applied at random positions in a network. Each time it is applied, the rule adds two edges to one face, and removes them from another. After many steps the pictures below show that faces with large numbers of edges appear. The average number of edges must always be 6 (see note above), but in a sufficiently large network the probability for a face to have n edges eventually approaches an equilibrium value of $8(n-2)(2n-3)!!(3/8)^n/n!$. (For large n this is approximately λ^n with $\lambda = 3/4$; if 1- and 2-edged regions are allowed then $\lambda = (3 + \sqrt{3})/6 \simeq 0.79$.) There may be some easy way to derive such results, but so far it has only been done using fairly sophisticated techniques from quantum field theory developed in the late 1970s. The starting point is to look at a

ϕ^3 field theory with SU(n) internal symmetry and to note that in the limit $n \to \infty$ what dominates are Feynman diagrams that have the structure of planar trivalent networks (see page 1040). And it then turns out that in zero spacetime dimensions the complete path integral for the theory can be evaluated exactly—yielding in effect a generating function for the number of possible networks. Parametric differentiation (to yield n-point correlation functions) then gives results for n-sided regions. Another result that has been derived is that the average total number $m[n]$ of edges of all faces around a given face with n edges is $7n + 3 + 9/(n + 1)$. Note that the networks obtained always have dimension 2 according to my definitions.





▪ **Cellular structures.** There are many systems in nature that consist of assemblies of discrete regions—and the lines that define the interfaces between these regions form networks. In many cases the regions are fixed once established (compare page 988). But in other cases there is continuing evolution, as for example in soap and other foams and froths, grains in metals and perhaps some biological tissues. In 2D situations the lines between regions generically form a trivalent planar network. In a soap foam, the geometrical layout of this network is determined by surface tension forces—with connections meeting at 120° at each node, though being slightly curved and of different lengths. Pressure differences lead to diffusion of gas and on average to von Neumann's Law that the area of an n-sided region changes linearly with time, at a rate proportional to $n - 6$. Typically the network topology of a foam continually rearranges itself through cascades of seemingly random T1 processes (rule (b) from page 511), with regions that reach zero size disappearing through T2 processes (reversed rule (a)). And as noted for example by Cyril Smith in the early 1950s there is a characteristic coarsening that occurs. Something similar is already visible in the pure T1 pictures in the note above. But results such as the so-called Aboav-Weaire law that $m[n]$ from the note above is in practice about $5n + c$ suggest that T2 processes are also important. (Processes like cell division in 2D biological tissue in effect directly add connections to a network. But this can again be thought of as a combination of T1 and T2 processes, and in appropriate idealizations can lead to very similar results.)

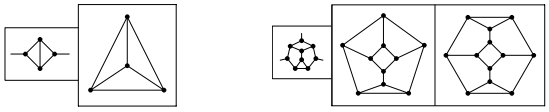
▪ **Page 514 · Cluster numbers.** The following tables give the total numbers of distinct clusters—with number of nodes going across the page, and number of dangling connections going down. (See also page 1038.)

	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	2	0	5	0	19
1	0	0	0	0	1	0	4	0	19	0
2	0	0	0	1	0	5	0	23	0	132
3	1	0	1	0	3	0	15	0	91	0
4	0	1	0	2	0	9	0	54	0	390
5	0	0	1	0	4	0	22	0	166	0
6	0	0	0	2	0	9	0	63	0	551

	1	2	3	4	5	6	7	8	9	10
7	0	0	0	0	2	0	17	0	157	0
8	0	0	0	0	0	4	0	38	0	424
9	0	0	0	0	0	0	6	0	80	0
10	0	0	0	0	0	0	0	11	0	180
11	0	0	0	0	0	0	0	0	18	0
12	0	0	0	0	0	0	0	0	0	37

▪ **Page 515 · Non-overlapping clusters.** The picture shows all distinct clusters with 3 dangling connections and 9 nodes that are not self-overlapping. The only smaller cluster with the same property is the trivial one with just a single node.

Most clusters that can overlap will be able to do so in an infinite number of possible networks. (One can see this by noting that they can overlap inside clusters with dangling connections, not just closed networks.) But there are some clusters that can overlap only in a few small networks. The pictures below show examples where this happens. The pictures in the main text still treat such clusters as non-overlapping.



If two clusters overlap, then this means that there is some network in which there are copies of these clusters that involve some of the same nodes. And it is possible to search for such a network by starting from a single node and then sequentially trying to take corresponding pieces from the two clusters.

▪ **1- and 2-connection clusters.** Clusters with just one or two dangling connections can always in effect be thought of just as adding extra structure to single connections in a network. But this extra structure can be important in the application of other rules—and can for example emulate something like having multiple colors of connections.

▪ **Connectedness.** It is not clear whether a network that represents the universe must remain globally connected, or whether pieces can break off. But any replacements that take connected clusters and yield connected clusters must always maintain the connectedness of any network.

■ **Reversibility.** By including both forward and backward versions of every transformation it is straightforward to set up reversible rules for network evolution. It is not clear, however, whether the basic rules for the universe are really reversible. It could well be that the apparent reversibility we see arises because the universe is effectively on an attractor, as discussed on page 1018. Note that if pieces of the universe can break off, but cannot reconnect, then there will inevitably be an irreversible loss of information.

■ **1/*n* expansion.** If there are n possible colors for each connection in a network, then for large n it turns out that the vast majority of networks will be planar. This idea was used in the 1980s as a way of simplifying the Feynman diagrams to consider in QCD and other quantum field theories. (See page 1039.)

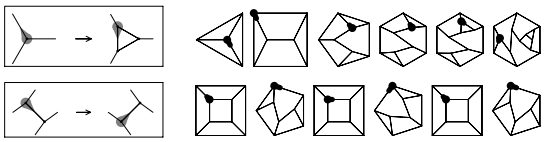
■ **Feynman diagrams.** In the standard approach to particle physics, possible interaction processes are represented by networks in which each node corresponds to an elementary interaction, and the nodes are joined by connections which correspond to the propagation of particles in spacetime. I can see no direct physical relationship between such diagrams and the networks I consider. However, at a mathematical level, the set of trivalent networks with n nodes formally corresponds to the set of n^{th} order Feynman diagrams in a ϕ^3 field theory. (Compare page 1039.)

■ **Chemical analogy.** The evolution of a network can be thought of as an idealized version of a chemical process in which molecules are networks of bonds. (See page 1193.)

■ **Symbolic representations.** Expressions in which common subexpressions are shared correspond to networks, as do collections of relations between objects representing nodes.

■ **Graph grammars.** The notion of generalizing substitutions for strings to the case of networks has been discussed in computer science since the 1960s—and a fair amount of formal work has been done on so-called graph grammars for specifying formal languages whose elements are networks. Even a good analog of regular languages has, however, not yet been found. But applications to constructing or verifying practical network-based system description schemes are quite often discussed. In mathematics rather little is usually done with anything but very trivial network substitutions. In mathematics, rather little is usually done with network substitutions, though the proof of the Four-Color Theorem in 1976 was for example based on showing that 300 or so possible replacement rules—if applied in an appropriate sequence—can transform any graph to have one of 1936 smaller subgraphs that require the same number of colors. (32 rules and 633 subgraphs are now known to be sufficient.)

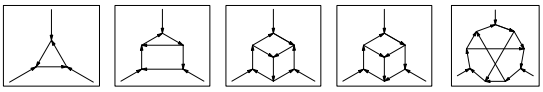
■ **Network mobile automata.** The analog of a mobile automaton can be defined for networks by setting up a single active node, then having rules which replace clusters of nodes around this active node, and move its position. The pictures below show two simple examples.



The total number of replacements that can be used in the rules of a network mobile automaton and which involve clusters with up to four nodes and have from 1 to 4 dangling connections is *{14, 10, 2727, 781}*. Despite looking at several hundred thousand cases I have not been able to find network mobile automata with especially complicated behavior.

Note that by having a cluster of nodes with a unique form it is possible to emulate a network mobile automaton using an ordinary network substitution system.

■ **Directed network systems.** If one adds directionality to the connections in a network it becomes particularly easy to set up rules for clusters of nodes that cannot overlap. For no two clusters whose dangling connections all point inwards can ever overlap, at least so long as neither of these clusters themselves contain subclusters whose dangling connections similarly all point inwards. The pictures below show a few examples of such clusters. Note that in a random network of n nodes, about $n/8$ such clusters typically occur.



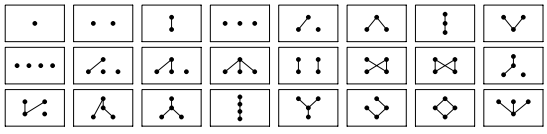
Space, Time and Relativity

■ **Page 516 · Posets.** The way I set things up, collections of events can be thought of as partially ordered sets (posets). If all events occurred in a definite sequence in time, this would define a total linear ordering for them. But with the setup I use, there is only a partial ordering of events, defined by causal connections. The causal networks I draw are so-called Hasse or order diagrams of the posets of events. If a connection goes directly from x to y in this network then x is said to cover y. And in general if there is a path from x to y then one writes $x > y$. The collection of all events that will lead to a given set of events (the union of their past light cones) is known as the filter of that set. Within a poset, there

can be sequences of elements that are totally ordered, and these are called chains. (The maximum length of any chain is sometimes called the dimension of a poset, but this is unrelated to the notions of dimension I consider.) There can also be sets of elements between which no ordering relations at all are defined, and these are called antichains.

Standard examples of posets include subsets of a set ordered by the subset relation, complex numbers ordered by magnitude, and integers ordered by divisibility. Posets first arose as general concepts in the late 1800s in connection with the development of mathematical logic, and to some extent abstract algebra. They became somewhat popular in the mid-1900s, both as formal generalizations in lattice theory, and as structures in various combinatorics applications. It was already noted in the 1920s that events in relativity theory formed posets.

The pictures below show the first few distinct possible Hasse diagrams for posets. For successive numbers of elements the total numbers of these are 1, 2, 5, 16, 63, 318, 2045, 16999, ...



▪ **Page 517 · Spacelike slices.** The definition of spacelike slices used here is directly analogous to what is used in traditional relativity theory (typically under names like spacelike hypersurfaces and Cauchy surfaces). There will normally be many different possible choices of spacelike slices, but in all cases a particular such slice is set up to represent what can consistently be thought of as all of space at a given time. One definition of a spacelike slice is then a maximal set of points in which no pair are causally related (corresponding to a maximal antichain in a poset). Another definition (equivalent for any connected causal network) is that spacelike slices are what consistently divide a causal network into a past and a future. And an intermediate definition is that a spacelike slice contains points that are not themselves causally related, but which appear in either the past or the future of every other point. Given a spacelike slice in a causal network, it is always possible to construct another such slice by finding all those points whose immediate predecessors are all included either in the original slice or its predecessors.

▪ **Page 518 · Speed of light.** In a vacuum the speed of light is 299,792,458 meters/second (and this is actually what is taken to define a meter). In materials light mostly travels slower—basically because there are delays when it is absorbed and reemitted by atoms. In a first approximation, the slowdown factor is the refractive index. But particularly in materials which can amplify light a whole sequence of peculiar effects have been observed—and it is fairly subtle to account correctly for incoming and outgoing signals, and to show that at least no energy or information is transmitted faster than c. The standard mathematical framework of relativity theory implies that any massless particle must propagate at c in a vacuum—so that not only light but also gravitational waves presumably go at this speed (and the same is at least approximately true of neutrinos). The effective mass for massive particles increases by a factor $1/\text{Sqrt}[1 - v^2/c^2]$ at speed v, making it take progressively more energy to increase v. At a formal mathematical level it is possible to imagine tachyons which always travel faster than c. But the structure of modern physics would find it difficult to accommodate interactions between these and ordinary particles.

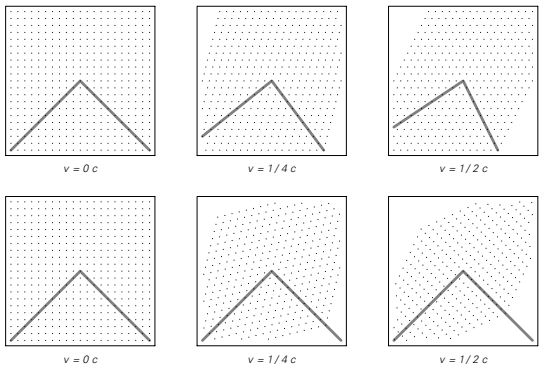
▪ **Page 522 · History of relativity.** (See also page 1028.) The idea that mechanical processes should work the same regardless of how fast one is moving was expressed by Galileo in the early 1600s, particularly in connection with the motion of the Earth—and was incorporated in the laws of mechanics formulated by Isaac Newton in 1687. But when the wave theory of light finally became popular in the mid-1800s it seemed to imply that no similar principle could be true for light. For it was generally assumed that waves of light must correspond to explicit disturbances in a medium or ether that fills space. And it was thus expected that for example the apparent speed of light would depend on how fast one was moving with respect to this ether. And indeed in particular this was what the equations for electromagnetism developed by James Maxwell in the 1860s seemed to suggest. But in 1881 an experiment by Albert Michelson (repeated more accurately in 1887 as the Michelson-Morley experiment and now done to the 10^{-20} level) showed that in fact this was not correct. Already in 1882 George FitzGerald and Hendrik Lorentz noted that if there was a contraction in length by a factor $\text{Sqrt}[1 - v^2/c^2]$ in any object moving at speed v (with c being the speed of light) then this would explain the result. And in 1904 Lorentz pointed out that Maxwell's equations are formally invariant under a so-called Lorentz transformation of space and time coordinates (see note below). Then in 1905 Albert Einstein proposed his so-called special theory of relativity—which took as its basic postulates not only that the laws of mechanics and electrodynamics are independent of how fast one is moving, but that this is also true of the speed of light.

And while at first these postulates might seem incompatible, what Einstein showed was that they are not—at least if modifications are made to the basic laws of mechanics. In the few years that followed, various formulations of this result were given, with Hermann Minkowski in 1908 showing that it could be derived if one just assumes that space and time enter all physical laws together in a certain kind of 4D vector. In the late 1800s Ernst Mach had emphasized the idea of formulating science and particularly mechanics in terms only of concepts that can actually be measured by observers. And in this framework Einstein and others gave what seemed to be almost purely deductive arguments for relativity theory—with the result that it generally came to be assumed that there was no meaningful sense in which one could ever imagine deriving relativity from anything more fundamental. Yet as I discussed earlier in the chapter, if a complete theory of physics is to be as simple as possible, then most things like relativity theory must in effect be derived from more basic features of the theory—as I start to try to do in the main text of this section.

■ **Standard treatment.** In a standard treatment of relativity theory one way to begin is to consider setting up a square grid of points in space and time—and then to ask what kind of transformed grid corresponds to this same set of points if one is moving at some velocity v. At first one might assume that the answer would just be a grid that has been sheared by the simple transformation $\{t, x\} \to \{t, x - v\,t\}$, as in the first row of pictures below. And indeed for purposes of Newtonian mechanics this so-called Galilean transformation is exactly what is needed. But as the pictures below illustrate, it implies that light cones tip as v increases, so that the apparent speed of light changes, and for example Maxwell's equations must change their form. But the key point is that with an appropriate transformation that affects both space and time, the speed of light can be left the same. The necessary transformation is the so-called Lorentz transformation

$$\{t, x\} \to \{t - v\,x/c^2,\ x - v\,t\}/\mathrm{Sqrt}[1 - v^2/c^2]$$

And from this the time dilation factor $1/\mathrm{Sqrt}[1 - v^2/c^2]$ shown on page 524 follows, as well as the length contraction factor $\mathrm{Sqrt}[1 - v^2/c^2]$. An important feature of the Lorentz transformation is that it preserves the quantity $c^2\,t^2 - x^2$—with the result that as v changes in the pictures below a given point in the grid traces out a hyperbola whose asymptotes lie on a light cone. Note that on a light cone $c^2\,t^2 - x^2$ always vanishes. Note also that the intersection of the past and future light cones for two events separated by a distance x in space and t in time always has a volume proportional exactly to $c^2\,t^2 - x^2$.



■ **Inferences from relativity.** The pictures on page 524 show that an idealized clock based on bouncing light between mirrors will exhibit relativistic time dilation. And from such derivations it is often assumed that the same result must hold for any possible clock system. But as a practical matter it does not. And indeed for example the clocks in GPS satellites are specifically set up so as to remove the effects of time dilation. And in the twin paradox one can certainly imagine that each twin could have an accelerometer whose readings they use to correct their clocks. Indeed, even when it comes to individual particles there are subtle effects associated with acceleration and radiation (see page 1062)—so that in the end not entirely clear that something like a biological system would actually in practice exhibit just standard time dilation.

One feature of relativity is that it implies that only relative motion is ultimately ever detectable. (This was also implied by Newtonian mechanics for purely mechanical systems.) And from this it is often concluded that there can be nothing like an ether that one can consider as defining an absolute state of rest in the universe. But in fact the cosmic microwave background in effect does exactly this. For in standard cosmological models it fills the universe, but is everywhere at rest relative to the global center of mass of the universe. And from the anisotropies we have observed in the microwave background it is thus possible to conclude that the Earth is moving at an absolute speed of about $c/10^3$ relative to the center of mass of the universe. In particle physics standard models also in effect introduce things that are assumed to be at rest relative to the center of mass of the universe. One example is the Higgs condensate discussed in connection with particle masses (see page 1047). Other possible examples include zero-point fluctuations in quantum fields.

Outside of science, relativity theory is sometimes given as evidence for various general ideas of cultural relativism (compare page 1131)—which have existed since well before

relativity theory in physics, and seem in the end to have no meaningful connection to it.

■ **Particle physics.** Relativity theory was originally formulated just for mechanics and electromagnetism. But its predictions like $E = m c^2$ were immediately applied for example to radioactivity, and soon it came to be assumed that the theory would work for any system at all—unless it involved gravity. So this has meant that in particle physics $c^2 t^2 - x^2 - y^2 - z^2$ is at some level the only quantity that ever appears. And to make mathematical work easier, what is very often done is to carry out the so-called Wick rotation $t \to i t$—so relativistic invariance is just independence on 4D orientation. (See page 1061.) But except in rather simple cases there is practically no evidence that results obtained after Wick rotation have anything to do with physical reality—and certainly the transformation removes some very basic phenomena such as particle propagation. One feature of it, however, is that it maps the equation for quantum mechanical time evolution into the equation for probabilities in statistical mechanics, with imaginary time corresponding to inverse temperature. And while it is conceivable that this mapping may have some deep significance, none has so far ever been identified.

■ **Time travel.** The idea that space and time are similar suggests that it might be possible to move backwards and forwards in time just like it is possible to move backwards and forwards in space. And indeed in the partial differential equations that define general relativity, it is formally possible for the motion of particles to achieve this, at least when there is sufficient negative energy density from matter or a cosmological constant. But even in this case there is no real progression in which one travels backwards in time. Instead, the possibility of motion that leads to earlier times simply implies a requirement of consistency between behavior at earlier and later times.

Elementary Particles

■ **Note for physicists.** My goal in the remainder of this chapter is not to present a specific ultimate model for physics, but rather to discuss at a fairly general level some features that I believe such a model will have, given the overall discoveries of this book, and the specific results I have described in this chapter. I am certainly aware that many physicists will want to know more details. But particularly in making contact with existing physics it is almost inevitable that all sorts of technical formalism will be needed—and to maintain balance in this book I have not included this here. (Given my own personal background in theoretical physics it will come as no surprise that I have often used such formalism in the process of working out what I describe in these sections.)

■ **Page 525 · Types of particles.** Current particle physics identifies three basic types of known elementary particles: leptons, quarks and gauge bosons. The known leptons are the electron (e), muon (μ) and tau lepton (τ), and their corresponding neutrinos (ν_e, ν_μ, ν_τ). Quarks exist inside hadrons like the proton and pion, but never seem to occur as ordinary free particles. Six types are known: u, d, c (charm), s (strange), t (top), b. Gauge bosons are associated with forces. Those currently known are the photon (γ) for electromagnetism (QED), W and Z for so-called weak interactions, and the gluon (g) for QCD interactions between quarks. Gravitons associated with gravitational forces presumably also exist. In ordinary matter, the only particles that contribute in direct ways to everyday physical, chemical and even nuclear properties are electrons, photons and effectively u and d quarks, and gluons. (These, together presumably with some type of neutrino, are the only types of particles that never seem to decay.) The first reasonably direct observations of the various types of particles were as follows (some were predicted in advance): e (1897), γ (~1905), u, d (1914/~1970), μ (1937), s (1946), ν_e (1956), ν_μ (1962), c (1974), τ, ν_τ (1975), b (1977), g (~1979), W (1983), Z (1983), t (1995).

Most particles exist in several variations. Apart from the photon (and graviton), all have distinct antiparticles. Each quark has 3 possible color configurations; the gluon has 8. Most particles also have multiple spin states. Quarks and leptons have spin 1/2, yielding 2 spin states (neutrinos could have only 1 if they were massless). Gauge bosons normally have spin 1 (the graviton would have spin 2) yielding 3 spin states for massive ones. Real massless ones such as the photon always have just 2. (See page 1046.)

In the Standard Model the idea of spontaneous symmetry breaking (see page 1047) allows particles with different masses to be viewed as manifestations of single particles, and this is effectively done for W, Z, γ, as well as for each of the 3 so-called families of quarks and leptons: u, d; c, s; t, b and e, ν_e; μ, ν_μ; τ, ν_τ. Grand unified models typically do this for all known gauge bosons (except gravitons) and for corresponding families of quarks and leptons—and inevitably imply the existence of various additional particles more massive than those known, but with properties that are somehow intermediate. Some models also unify different families, and supersymmetric models unify quarks and leptons with gauge bosons.

■ **History.** The idea that matter—and light—might be made up of discrete particles was already discussed in antiquity

(see page 876). But it was only in the mid-1800s that there started to be real evidence for the existence of some kind of discrete atoms of matter. Yet at the time, the idea of fields was popular, and it was believed that the universe must be filled with a continuous fluid-like ether responsible at least for light and other electromagnetic phenomena. So for example following ideas of William Rankine from 1849 William Thomson (Kelvin) in 1867 suggested that perhaps atoms might be like knotted stable vortex rings in the ether—with different knots corresponding to different chemical elements. But though it initiated the mathematical classification of knots, and now has certain conceptual similarities to what I discuss in this book, the details of this model did not work out—and it had been largely abandoned even before the electron was discovered in 1897. Ernest Rutherford's work in the 1910s on scattering from atoms introduced the idea of an atomic nucleus, and after the discovery of the neutron in 1932 it became clear that the main constituents of nuclei were protons and neutrons. The positron and the muon were discovered in cosmic rays in the 1930s, followed in the 1940s by a handful of other particles. By the 1960s particle accelerators were finding large numbers of new particles every year. And the hypothesis was then suggested that all these particles might actually be composed of just three more fundamental particles that became known as quarks. An alternative so-called democratic or bootstrap hypothesis was also suggested: that somehow any particle could just be viewed as a composite of all others with the same overall properties—with everything being determined by consistency in the web of interactions between particles, and no particles in a sense being more fundamental than others. But by the early 1970s experiments on so-called deep inelastic scattering had given increasingly direct evidence for point-like constituents inside particles like protons—and by the mid-1970s these were routinely identified with quarks.

As soon as the electron was discovered there were questions about its possible size. For if its charge was distributed over a sphere of radius r, this was expected to lead to electrostatic repulsion energy proportional to $1/r$. And although it was suggested around 1900 that effects associated with this might account for the mass of the electron, this ran into problems with relativity theory, and it also remained mysterious just what might hold the electron together. (A late suggestion made in 1953 by Hendrik Casimir was that it could be forces associated with zero-point fluctuations in quantum fields—but at least with the simplest setup these turned out to have wrong sign.)

The development of quantum theory in the 1920s showed that discrete particles will inevitably exhibit continuous wave-like features in their spatial distribution of probability amplitudes. But traditional quantum mechanics and quantum field theory are both normally formulated with the assumption that the basic particles they describe have zero intrinsic spatial size. Sometimes nonzero size is taken into account by inserting additional interaction parameters—as done in the 1950s with magnetic moments and form factors of protons and neutrons. But for example in quantum electrodynamics the definite assumption is made that electrons are intrinsically of zero size. Quantum fluctuations make any particle in an interacting field theory effectively be surrounded by virtual particles. Yet not unlike in classical electrodynamics having zero intrinsic size for the electron still immediately suggests that an electron should have infinite self-energy. In the 1930s ideas about avoiding this centered around modifying basic laws of electrodynamics or the structure of spacetime (see page 1027). But the development of renormalization in the 1940s showed that these infinities could in effect just be factored out. And by the 1960s a long series of successes in the predictions of QED had led to the almost universal belief that its assumption of point-like electrons must be correct. It was occasionally suggested that the muon might be some kind of composite object. But experiments seemed to indicate that it was in every way identical to the electron, except in mass. And although no reasonable explanation for its existence was found, it came to be generally assumed by the 1970s that it was just another point-like particle. And indeed—apart from few rare suggestions to the contrary—the same is now assumed throughout mainstream practical particle physics for all of the basic particles that appear in the Standard Model. (Actual experiments based on high-energy scattering and precision magnetic moment measurements have shown only that electrons and muons must have sizes smaller than about $\hbar\, c/(10\ \mathrm{TeV}) \simeq 10^{-20}\ m$—or about 10^{-5} times the size of a proton. One can make arguments that composite particles this small should have masses much larger than are observed—but it is easy to find theories that avoid these.)

In the 1980s superstring theory introduced the idea that particles might actually be tiny 1D strings—with different types of particles corresponding essentially just to strings in different modes of vibration. Since the 1960s it has been noted in many simplified quantum field theories that there can be a kind of duality in which a soliton or other extended field configuration in one representation becomes what acts like an elementary particle in another representation. And in the late 1990s there were indications that such phenomena could occur in generalized string theories—leading to suggestions of at least an abstract correspondence between

for example particles like electrons and gravitational configurations like black holes.

■ **Page 526 · Topological defects.** An idealized vortex in a 2D fluid involves velocity vectors that in effect wind around a point—and can never be unwound by making a series of small local perturbations. The result is a certain kind of stability that can be viewed as being of topological origin. One can classify forms of stability like this in terms of the mathematics of homotopy. Most common are point and line defects in vector fields, but more complicated defects can occur, notably in liquid crystals, models of condensates in the early universe, and certain nonlinear field theories. Analogs of homotopy can presumably be devised to represent certain forms of stability in systems like the networks I consider.

■ **Page 527 · Kuratowski's theorem.** Any network can be laid out in 3D space. (This is related to the Whitney embedding theorem that any d-dimensional manifold can be embedded in $(2d+1)$-dimensional space.) When one says that a network is planar what one means is that it can be laid out in ordinary 2D space without any lines crossing. Kuratowski's theorem that planarity is associated with the absence of specific subgraphs in a network is an important result in graph theory established in the late 1920s. A subgraph is formally defined to be what one gets by selecting just some subset of connections in a network—and with this definition Kuratowski's theorem must allow extensions of K_5 and $K_{3,3}$ where extra nodes have been inserted in the middle of connections. (K_5 and $K_{3,3}$ are examples of so-called complete graphs, obtained by taking sets of specified numbers of nodes and connecting them in all possible ways.) Another approach is to consider reducing whole networks to so-called minors by deleting connections or merging connected nodes, and in this case Wagner's theorem shows that any non-planar network must be exactly reducible to either K_5 or $K_{3,3}$.

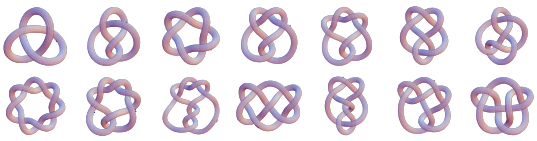
One can generalize the question of planarity to asking whether networks can be laid out on 2D surfaces with various topological structures—and in fact the genus of a graph can be defined to be the number of handles that must be added to a plane to embed the graph without crossings. But even on a torus it turns out that there is no finite set of (extended) subgraphs whose absence guarantees that a network can successfully be laid out. Nevertheless, if one considers minors a finite list does suffice—though for example on a torus it is known that at least 800 (and perhaps vastly more) are needed. (There is in fact a general theorem established since the 1980s that absolutely any list of networks—say for example ones that cannot be laid on a given surface—must actually in effect always all be reducible to some finite list of minors.) Note that finding the genus for a particular trivalent network is in general NP-complete.

■ **Page 527 · Gauge invariance.** It is often convenient to define quantities for which only differences or derivatives matter. In classical physics an example is electric potential, which can be shifted by any constant amount without affecting voltage differences or the electric field given by its gradient. In the mid-1800s the idea emerged of a vector potential whose curl gives the magnetic field, and it was soon recognized—notably by James Clerk Maxwell—that any function whose curl vanishes (and that can therefore normally be written as a gradient) could be added to the vector potential without affecting the magnetic field. By the end of the 1800s the general conditions on electromagnetic potentials for invariance of fields were known, though were not thought particularly significant. In 1918 Hermann Weyl tried to reproduce electromagnetism by adding the notion of an arbitrary scale or gauge to the metric of general relativity (see page 1028)—and noted the "gauge invariance" of his theory under simultaneous transformation of electromagnetic potentials and multiplication of the metric by a position-dependent factor. Following the introduction of the Schrödinger equation in quantum mechanics in 1926 it was almost immediately noticed that the equations for a charged particle in an electromagnetic field were invariant under gauge transformations in which the wave function was multiplied by a position-dependent phase factor. The idea then arose that perhaps some kind of gauge invariance could also be used as the basis for formulating theories of forces other than electromagnetism. And after a few earlier attempts, Yang-Mills theories were introduced in 1954 by extending the notion of a phase factor to an element of an arbitrary non-Abelian group. In the 1970s the Standard Model then emerged, based entirely on such theories. In mathematical terms, gauge theories can be viewed as describing fiber bundles in which connections between values of group elements in fibers at neighboring spacetime points are specified by gauge potentials—and curvatures correspond to gauge fields. (General relativity is in effect a special case in which the group elements are themselves related to spacetime coordinates.)

■ **Page 527 · Identifying particles.** In something like a class 4 cellular automaton it is quite straightforward to start enumerating possible persistent structures—as we saw in Chapter 6. But in a network system it can be much more difficult. Ultimately what one wants to do is to find what possible types of forms for local regions are inequivalent under the application of the underlying rules. But in general it may be undecidable even whether two such forms are actually equivalent (compare the notes below and on page

1051)—since to tell this one might need to be able to apply the rules infinitely many times. In specific cases, however, generalizations of concepts like planarity and homotopy may provide useful guides. And a first step may be to look at small closed networks and try to determine which of these can be transformed into each other by a given set of rules.

■ **Knot theory.** Somewhat analogous to the problem in the note above is the problem of classifying knots. The pictures below show some of the simplest distinct knots. But given presentations of two knots, no finite procedure is known that determines in general whether the knots are equivalent (or constructs a sequence of Reidemeister moves that transform one into the other). Quite probably this is in general undecidable, though since the 1920s a few polynomial invariants have been discovered—with recent ones being related to ideas from quantum field theory—that have allowed some progress to be made. (Even the problem of determining whether a knot specified by line segments is trivial is known to be NP-complete.)



■ **Page 528 · Charge quantization.** It is an observed fact that the electric and other charges of all particles are simple rational multiples of each other. In the context of electromagnetism alone, there would be no particular reason to expect this (unless magnetic monopoles exist). But as soon as different particles are related by a non-Abelian symmetry group, then the discreteness of the representations of such a group immediately implies that all charges must be rational multiples of each other.

■ **Spin.** Even when they appear to be of zero size, particles exhibit intrinsic angular momentum known as spin. The total spin is always a fixed multiple of the basic unit $\hbar$: 1/2 for quarks and leptons, 1 for photons and other ordinary gauge bosons, 2 for gravitons, and in theory 0 for Higgs particles. (Observed mesons have spins up to perhaps 5 and nuclei up to more than 50.) Particles of higher spin in effect require more information to specify their orientation (or polarization or its analog). And in the context of network models it could be that spin is somehow related to something as simple as the number of places at which the core of a particle is attached to the rest of the network. Spin values can be thought of as specifying which irreducible representation of the group of symmetries of spacetime is needed to describe a particle after momentum has been factored out. For ordinary massive particles in d-dimensional space the group is Spin(d), while for massless particles it is $E(d-1)$ (the Euclidean group). (For tachyons, it would be fundamentally non-compact, forcing continuous spin values.) For small transformations, Spin(d) is just the ordinary rotation group SO(d), but globally it is its universal cover, or SU(2) in 3D. And this can be thought of as what allows half-integer spins, which must be described by spinors rather than vectors or tensors. Such objects have the property that they are not left invariant by 360 ° rotations, but only by 720 ° ones—a feature potentially fairly easy to reproduce with networks, perhaps even without definite integer dimensions. In the standard formalism of quantum field theory it can be shown that (above 2D) half-integer spins must always be associated with fermions (which for example satisfy the exclusion principle), and integer spins with bosons. (This spin-statistics connection also seems to hold for various kinds of objects defined by extended field configurations.)

■ **Page 528 · Particle masses.** The measured masses of known elementary particles in units of GeV (roughly equal to the proton mass) are: photon: 0, electron: 0.000510998902; muon: 0.1056583569; τ lepton: 1.77705; W: 80.4; Z: 91.19. Recent evidence suggests a mass of about 10^{-11} GeV for at least one type of neutrino. Quarks and gluons presumably never occur as free particles, but still act in many ways as if they have definite masses. For all of them their confinement contributes perhaps 0.3 GeV of effective mass. Then there is also a direct mass: gluons 0; u: ~0.005; d ~0.01; s: ~0.2; c: 1.3; b: 4.4; t: 176 GeV. Note that among sets of particles that have the same quantum numbers—like d, s, b or γ, Z—mixing occurs that makes states of definite mass—that would propagate unchanged as free particles—differ by a unitary transformation from states that are left unchanged by interactions. When one sets up a quantum field theory one can typically in effect insert various mass parameters for particles. Self-interactions normally introduce formally infinite corrections—but if a theory is renormalizable then this means that there are only a limited number of independent such corrections, with the result that relations between masses of different particles are preserved. In quantum field theory any particle is always surrounded by a kind of cloud of virtual particles interacting with it. And following the Uncertainty Principle phenomena involving larger momentum scales will then to probe progressively smaller parts of this cloud—yielding different effective masses. (The masses tend to go up or down logarithmically with momentum scale—following so-called renormalization group equations.)

The Standard Model starts off with certain symmetries that force the masses of all ordinary particles to be zero. But then one assumes that nonzero masses are generated by spontaneous symmetry breaking. One starts by taking each particle to be coupled to a so-called Higgs field. Then one introduces self-interactions in this field so as to make its stable state be one that has constant nonzero value throughout the universe. But this means that as particles propagate, their interactions with the background give them an effective mass. And by having Higgs couplings be proportional to observed particle masses, it becomes inevitable that these will be the masses of particles. One prediction of the usual version of this mechanism for mass is that a definite Higgs particle should exist—which in the minimal Standard Model experiments should observe fairly soon. At times there have been hopes of so-called dynamical symmetry breaking giving the same effective results as the Higgs mechanism, but without an explicit Higgs field—perhaps through something similar to various phenomena in condensed matter physics. String theory, like the Standard Model, tends to start with zero mass particles—and then hopes that an appropriate Higgs-like mechanism will generate nonzero ones.

▪ **More particles.** To produce more massive particles requires higher-energy particle collisions, and today's accelerators only allow one to search up to masses of perhaps 200 GeV. (Sufficiently stable particles could have survived from the early universe, and a few cosmic ray interactions in principle give higher energies—but are normally too rare to be useful.) I am not sure whether in my approach one should expect an infinite series of progressively more massive particles. The example of nonplanarity might suggest not, but even in the class 4 cellular automata discussed in Chapter 6 it is not clear whether fundamentally different progressively larger structures will appear forever. In quantum field theory particles of any mass can always in principle exist for short times in virtual form. But normally their effects decrease like powers of their mass—making them hard to measure. In two kinds of cases, however, this does not happen: one is so-called anomalies, the other interactions with the Higgs field, in which couplings are proportional to mass. In the minimal Standard Model it turns out to be impossible to get quarks or leptons with masses much above about 200 GeV without destabilizing the vacuum (a fact pointed out by David Politzer and me in 1979). But with more complicated models one can avoid this constraint. In supersymmetric models—and string theory—there are typically also all sorts of other types of particles, assumed to have high masses since they have not been observed. There is evidence against any more than the three known generations of quarks and leptons in that the decay process $Z^0 \to \nu\,\bar{\nu}$ has a rate that rather accurately agrees with what is expected from just three types of low-mass neutrinos.

▪ **Page 530 · Expansion of the universe.** See page 1055.

The Phenomenon of Gravity

▪ **History.** With the Earth believed to be the center of the universe, gravity did not seem to require much explanation: it was just a force bringing things to a natural place. But with the advent of Copernican astronomy in the 1500s something more was needed. In the early 1600s Galileo noted that the force of gravity seems to depend only on the mass of an object, and not on any of its other features. In 1687 Isaac Newton then suggested a universal inverse square law of gravity between objects. In the 1700s and 1800s all sorts of celestial mechanics was done on the basis of this—with occasional observational anomalies being resolved for example by the discovery of new planets. Starting in the mid-1800s there were attempts to formulate gravity in the same way as electromagnetism—and in 1900 it was for example suggested that gravitational effects might propagate at the speed of light. Following his introduction of relativity theory in 1905, Albert Einstein began to seek a theory of gravity that would fit in with it. Ordinary special relativity has the feature that it assumes that systems behave the same regardless of their overall velocity—but not regardless of their acceleration. In 1907 Einstein then suggested the equivalence principle that gravity always locally has the same effect as an acceleration. (This principle requires only slightly more than Galileo's idea of the equivalence of gravitational and inertial mass, which has now been verified to the 10^{-12} level.) But by 1912 Einstein realized that if the effective laws of physics were somehow to remain the same in systems with different accelerations (or in different gravitational fields) then this would require a change in their perceived geometry. And building on ideas of differential geometry and tensor calculus from the late 1800s Einstein then began to formulate the concept that gravity is associated with curvature of space. In the late 1800s Ernst Mach had argued that phenomena like acceleration and rotation could ultimately be defined only relative to matter in the universe. And partly on this basis Einstein used the idea that curvature in space must be like a field produced by matter—leading eventually to his formulation in 1915 of the standard Einstein equations for general relativity. An immediate prediction of these was a deviation from the inverse square law, explaining an observed precession in the orbit of Mercury. After a dramatic verification in 1919 of predicted bending of light by

the Sun, general relativity began to be widely accepted. In the 1920s expansion of the universe was discovered, and this was seen to be consistent with general relativity. In the 1940s study of the evolution of stars then led to discussion of what became known as black holes. But for the most part general relativity was still viewed as being highly elegant though of little practical relevance. In the 1960s, however, more work began to be done on it. The discovery of the cosmic microwave background in 1965 led to increasing interest in cosmology. Precision tests—particularly with spacecraft—were designed. In calculations it was sometimes difficult to tell what was a genuine effect, and what was just a feature of the particular coordinates used. But a variety of increasingly abstract mathematical methods were developed, leading notably to general theorems about inevitability of singularities. Detailed calculations tended to require complicated symbolic tensor manipulation (with some associated problems being NP-complete), but with the development of computer algebra this gradually became more feasible—and by the mid-1970s approximate numerical methods were also being used. Various alternative formulations of general relativity were proposed, based for example on tetrads, spinors and twistors (and more recently on connection, loop and non-commutative geometry methods)—but none led to any great simplification. Meanwhile, there continued to be ever more accurate experimental tests of general relativity in the solar system—and at least in the weak gravitational fields available there (with metrics differing from the identity by at most one part in 10^6), all have worked out to around the 10^{-3} level. Starting in the 1960s, more and more ambitious gravitational wave detectors have been built—although none as yet have actually observed anything. Measurements done on a binary pulsar system are nevertheless consistent at a 10^{-3} level with the emission of gravitational radiation in a fairly strong gravitational field at the rate implied by general relativity. And since the 1980s there has been increasing conviction that at least indirect effects of black holes associated with very strong gravitational fields are being observed.

Over the years, some variants of general relativity have been proposed. At least when formulated in terms of tensors, none have quite the simplicity of the original theory—but some lead to rather different predictions, such as an absence of singularities like black holes. Ever since quantum theory began in the early 1900s there has been discussion of quantum gravity—and almost every major method developed for handling other quantum phenomena has been tried on gravity. Starting in the 1980s a variety of methods more specific to quantum gravity were also pursued, but none have yet had convincing success. (See page 1054.)

▪ **Differential geometry.** Standard descriptions of properties like curvature—as used for example in general relativity—are normally based on differential geometry. In its usual formulation this assumes that space is continuous, and can always effectively be treated as some kind of deformed version of ordinary Euclidean space—thus forming what is known as a manifold. The result of this is that points in space can always be specified by lists of coordinates—although historically one of the objectives of differential geometry has been to find ways to define properties like curvature so that they do not depend on the choice of such coordinates. The geometrical properties of a space are in general specified by its so-called metric—and this metric allows one to compute quantities based on lengths and angles from coordinates. The metric can be written as a matrix g, defined so that the analog for infinitesimal vectors u and v of $u \cdot v$ in ordinary Euclidean space is $u \cdot g \cdot v$. (This is essentially equivalent to saying that infinitesimal arc length is related to infinitesimal coordinate distances by $ds^2 = g_{i,j}\, dx_i\, dx_j$.) In d dimensions the metric g for a so-called Riemannian space can in general be any $d \times d$ positive-definite symmetric matrix—and can vary with position. But for ordinary flat Euclidean space it is always just *IdentityMatrix[d]* (at least with Cartesian coordinates). Within say a surface whose points $\{x_1, x_2, \ldots\}$ are obtained by evaluating an expression e as a function of parameters p (so that for example $e = \{x, y, f[x, y]\}$, $p = \{x, y\}$ for a *Plot3D* surface) the metric turns out to be given by

```
(Transpose[#] . # &)[Outer[D, e, p]]
```

In ordinary Euclidean space a defining feature of geometry is that the shortest path between two points is a straight line. But in an arbitrary space things can be more complicated, and in general such a path will be a geodesic (see note below) which can have a more complicated form. If the coordinates along a path are given by an expression s (such as $\{t, 1 + t, t^2\}$) that depends on a parameter t, and the metric at position p is $g[p]$, then the length of a path turns out to be

```
Integrate[Sqrt[∂t s . g[s] . ∂t s], {t, t1, t2}]
```

and geodesics then correspond to paths that extremize this quantity. In ordinary Euclidean space, such paths are straight lines, so that the length of a path between points with lists of coordinates a and b is just the ordinary Euclidean distance *Sqrt[(a - b) . (a - b)]*. But in general, even though geodesics are not straight lines their lengths can still be used to define a so-called geodesic distance—which turns out to have all the various properties of a distance discussed on page 1030.

If one draws a circle of radius r on a page, then the smaller r is, the more curved the circle will be—and one can define the

circle to have a constant curvature equal to $1/r$. If one draws a more general curve on a page, one can define its curvature at every point by seeing what size of circle fits it best at that point—or equivalently what the coefficients are in a quadratic approximation. (Compare page 418.) With a 2D surface in ordinary 3D space, one can imagine fitting quadrics (generalized ellipsoids). But these are now specified by two radii, yielding two principal curvatures. And in general these curvatures depend on the way the surface is laid out in 3D space. But a crucial point noted by Carl-Friedrich Gauss in the 1820s is that the product of such curvatures—the so-called Gaussian curvature—is always independent of how the surface is laid out, and can thus be viewed as intrinsic to the surface itself, and for example determined purely from the metric for the 2D space corresponding to the surface.

In a 2D space, intrinsic curvature is completely specified just by Gaussian curvature. In higher-dimensional spaces, there are more components, but in general they are all part of the so-called Riemann tensor—a rank-4 tensor introduced by Bernhard Riemann in 1854. (In *Mathematica*, the explicit form of such a tensor can be represented as a nested list for which `TensorRank[list] == 4`.) Several descriptions of the Riemann tensor can be given. One is based on looking at infinitesimal vectors u, v and w and asking how much w differs when transported two ways around the edges of a parallelogram, from x to $x+u+v$ via $x+u$ and via $x+v$. In ordinary flat space there is no difference, but in general the difference is a vector that is defined to be `Riemann . u . v . w`. (The `Riemann` that appears here is formally R_{ijk}^{l}.) Another description of the Riemann tensor is based on geodesics. In flat Euclidean space any two geodesics that start parallel always remain so. But a defining feature of general non-Euclidean spaces is that this is not in general so. And it turns out that the Riemann tensor is what determines the rate at which geodesics deviate from being parallel. Still another description of the Riemann tensor is as the coefficient of the quadratic terms in an expansion of the metric about a particular point, using so-called normal coordinates set up to make linear terms vanish. In general the Riemann tensor can always be computed from the metric, though it is somewhat complicated. If p is a list of coordinate parameters that appear in a d-dimensional metric g, then

```
Riemann = Table[∂p[[j]] Γ[[i, k]] - ∂p[[i]] Γ[[j, k]] +
    Γ[[i, k]] . Γ[[j]] - Γ[[j, k]] . Γ[[i]], {i, d}, {j, d}, {k, d}]
```

where the so-called Christoffel symbol Γ_{ij}^{k} is

```
Γ = With[{gi = Inverse[g]}, Table[Sum[
      gi[[l, k]] (∂p[[j]] g[[i, l]] + ∂p[[i]] g[[j, l]] - ∂p[[l]] g[[i, j]]),
      {l, d}], {i, d}, {j, d}, {k, d}]]/2
```

There are d^4 elements in the nested lists for `Riemann`, but symmetries and the so-called Bianchi identity reduce the number of independent components to $1/12\, d^2\, (d^2-1)$—or 20 for $d=4$. One can then compute the Ricci tensor ($R_{ik} = R_{ijk}^{\ j}$) using

```
RicciTensor = Map[Tr, Transpose[Riemann, {1, 3, 2, 4}], {2}]
```

and this has $1/2\, d\, (d+1)$ independent components in $d>2$ dimensions. (The parts of the Riemann tensor not captured by the Ricci tensor correspond to the so-called Weyl tensor; for $d=2$ the Ricci tensor has only one independent component, equal to the negative of the Gaussian curvature.) Finally, the Ricci scalar curvature is given by

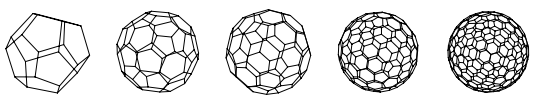
```
RicciScalar = Tr[RicciTensor . Inverse[g]]
```

▪ **Page 531 · Geodesics.** On a sphere all geodesics are arcs of great circles. On a surface of constant negative curvature (like (c)) geodesics diverge exponentially, as noted in early work on chaos theory (see page 971). The path of a geodesic can in general be found by requiring that the analog of acceleration vanishes for it. In the case of a surface defined by `z == f[x, y]` this is equivalent to solving

```
x''[t] == -(f^(1,0)[x[t], y[t]] (y'[t]^2 f^(0,2)[x[t], y[t]] +
     2 x'[t] y'[t] f^(1,1)[x[t], y[t]] + x'[t]^2 f^(2,0)[x[t], y[t]]))/
  (1 + f^(0,1)[x[t], y[t]]^2 + f^(1,0)[x[t], y[t]]^2)
```

together with the corresponding equation for y'', as already noted by Leonhard Euler in 1728 in connection with his development of the calculus of variations.

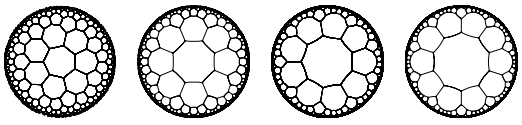
▪ **Page 532 · Spherical networks.** One can construct networks of constant positive curvature by approximating the surface of a sphere—starting with a dodecahedron and adding hexagons. (Euler's theorem implies that at any stage there must always be exactly 12 pentagonal faces.) The following are examples with 20, 60, 80, 180 and 320 nodes:



The object with 60 nodes is a truncated icosahedron—the shape of a standard soccer ball, as well the shape of the fullerene molecule C_{60}. (Note that in C_{60} one of the connections at each node is always a double chemical bond, since carbon has valence 4.) Geodesic domes are typically duals of such networks—with three edges on each face.

▪ **Hyperbolic networks.** Any surface that always has positive curvature must eventually close up to form something like a sphere. But a surface that has negative curvature (and no holes) must in some sense be infinite—more like cases (c) and (d) on page 412. Yet even in such a case one can always define coordinates that nominally allow the surface to be drawn in a finite way—and the Poincaré disk model used in the pictures below is the standard way of doing this. In ordinary flat space, regular polygons with more than 6

sides can never form a tessellation. But in a space with negative curvature this is possible for polygons with arbitrarily many sides—and the networks that result have been much studied as Cayley graphs of Fuchsian groups. One feature of these networks is that the number of nodes reached in them by following r connections always grows like 2^r. But if one intersperses hexagons in the networks (as in the main text) then one finds that for small r the number of nodes just grows like r^2—as one would expect for something like a 2D surface. But if one tries to look at growth rates on scales that are not small compared to characteristic lengths associated with curvature then one again sees exponential growth—just as in the case of a uniform tessellation without hexagons.



▪ **Page 533 · Sphere volumes.** In ordinary flat Euclidean space the area of a 2D circle is πr^2, and the volume of a 3D sphere $4 \pi r^3/3$. In general, the volume of a sphere in d-dimensional Euclidean space is $s[d]\, r^d$ where $s[d] = \pi^{d/2}/(d/2)!$ (the surface area is $d\, s[d]\, r^{d-1}$). (The function $s[d]$ has a maximum around $d = 5.26$, then decreases rapidly with d.)

If instead of flat space one considers a space defined by the surface of a 3D sphere—say with radius a—one can ask about areas of circles in this space. Such circles are no longer flat, but instead are like caps on the sphere—with a circle of radius r containing all points that are geodesic (great circle) distance less than r from its center. Such a circle has area

$$2 \pi a^2 (1 - \text{Cos}[r/a]) = \pi r^2 (1 - r^2/(12 a^2) + r^4/(360 a^4) - \ldots)$$

In the d-dimensional space corresponding to the surface of a $(d+1)$-dimensional sphere of radius a, the volume of a d-dimensional sphere of radius r is similarly given by

$$d\, s[d]\, a^d \text{Integrate}[\text{Sin}[\theta]^{d-1}, \{\theta, 0, r/a\}] = s[d]\, r^d (1 - d (d-1) r^2/((6 (d+2)) a^2) + (d (5 d^2 - 12 d + 7)) r^4/((360 (d+4)) a^4) + \ldots)$$

where

$$\text{Integrate}[\text{Sin}[x]^{d-1}, x] = -\text{Cos}[x]\ \text{Hypergeometric2F1}[1/2, (2-d)/2, 3/2, \text{Cos}[x]^2]$$

In an arbitrary d-dimensional space the volume of a sphere can depend on position, but in general it is given by

$$s[d]\, r^d (1 - \text{RicciScalar}\, r^2/(6 (d+2)) + \ldots)$$

where the Ricci scalar curvature is evaluated at the position of the sphere. (The space corresponding to a $(d+1)$-dimensional sphere has $\text{RicciScalar} = d (d-1)/a^2$.) The $d = 2$ version of this formula was derived in 1848; the general case in 1917 and 1939. Various derivations can be given. One can start from the fact that the volume density in any space is given in terms of the metric by $\text{Sqrt}[\text{Det}[g]]$. But in normal coordinates the first non-trivial term in the expansion of the metric is proportional to the Riemann tensor, yet the symmetry of a spherical volume makes it inevitable that the Ricci scalar is the only combination of components that can appear at lowest order. To next order the result is

$$s[d]\, r^d (1 - \text{RicciScalar}\, r^2/(6 (d+2)) + (5\, \text{RicciScalar}^2 - 3\, \text{RiemannNorm} + 8\, \text{RicciNorm} - 18\, \text{Laplacian}[\text{RicciScalar}])\, r^4/(360 (d+2) (d+4)) + \ldots)$$

where the new quantities involved are

```
RicciNorm = Norm[RicciTensor, {g, g}]

RiemannNorm = Norm[Riemann, {g, g, g, Inverse[g]}]

Norm[t_, gl_] := Tr[Flatten[t Dual[t, gl]]]

Dual[t_, gl_] := Fold[Transpose[#1 . Inverse[#2], RotateLeft[
                    Range[TensorRank[t]]]] &, t, Reverse[gl]]

Laplacian[f_] := Inner[D, Sqrt[Det[g]]
                   (Inverse[g] . Map[∂# f &, p]), p]/Sqrt[Det[g]]
```

In general the series in r may not converge, but it is known that at least in most cases only flat space can give a result that shows no correction to the basic r^d form. It is also known that if the Ricci tensor is non-negative, then the volume never grows faster than r^d.

▪ **Cylinder volumes.** In any d-dimensional space, the volume of a cylinder of length x and radius r whose direction is defined by a unit vector v turns out to be given by

$$s[d-1]\, r^{d-1}\, x\ (1 - (d-1) (\text{RicciScalar} - \text{RicciTensor} \,.\, v \,.\, v)\, r^2/(d+1) + \ldots)$$

Note that what determines the volume of the cylinder is curvature orthogonal to its direction—and this is what leads to the combination of Ricci scalar and tensor that appears.

▪ **Page 533 · Discrete spaces.** Most work with surfaces done on computers—whether for computer graphics, computer-aided design, solving boundary value problems or otherwise—makes use of discrete approximations. Typically surfaces are represented by collections of patches—with a simple mesh of triangles often being used. The triangles are however normally specified not so much by their network of connections as by the explicit coordinates of their vertices. And while there are various triangulation methods that for example avoid triangles with small angles, no standard method yields networks analogous to the ones I consider in which all triangle edges are effectively the same length.

In pure mathematics a basic idea in topology has been to look for finite or discrete ways to capture essential features of continuous surfaces and spaces. And as an early part of this Henri Poincaré in the 1890s introduced the concept of approximating manifolds by cell complexes consisting of collections of generalized polyhedra. By the 1920s there was

then extensive work on so-called combinatorial topology, in which spaces are thought of as being decomposed into abstract complexes consisting say of triangles, tetrahedra and higher-dimensional simplices. But while explicit coordinates and lengths are not usually discussed, it is still imagined that one knows more information than in the networks I consider: not only how vertices are connected by edges, but also how edges are arranged around faces, faces around volumes, and so on. And while in 2D and 3D it is possible to set up such an approximation to any manifold in this way, it turns out that at least in 5D and above it is not. Before the 1960s it had been hoped that in accordance with the Hauptvermutung of combinatorial topology it would be possible to tell whether a continuous mapping and thus topological equivalence exists between manifolds just by seeing whether subdivisions of simplicial complexes for them could be identical. But in the 1960s it was discovered that at least in 5D and above this will not always work. And largely as a result of this, there has tended to be less interest in ideas like simplicial complexes.

And indeed a crucial point for my discussion in the main text is that in formulating general relativity one actually does not appear to need all the structure of a simplicial complex. In fact, the only features of manifolds that ultimately seem relevant are ones that in appropriate limits are determined just from the connectivity of networks. The details of the limits are mathematically somewhat intricate (compare page 1030), but the basic approach is straightforward. One can find the volume of a sphere (geodesic ball) in a network just by counting the number of nodes out to a given network distance from a certain node. And from the limiting growth rate of this one can immediately get the Ricci scalar curvature—just as in the continuous case discussed above. To get the Ricci tensor one also needs a direction. But one can get this from a geodesic—which is in effect the analog of a straight line in the network. Note that unlike in a continuous space there is however usually no obvious way to continue a geodesic in a network. And in general, some—but not all—of the standard constructions used in continuous spaces can also immediately be used in networks. So for example it is straightforward to construct a triangle in a network: one just starts from a particular node, follows geodesics to two others, then joins these with a geodesic. But to extend the triangle into a parallelogram is not so easy—since there is no immediate notion of parallelism in the network. And this means that neither the Riemann tensor, nor a so-called Schild ladder for parallel transport, can readily be constructed.

Since the 1980s there has been increasing interest in formulating notions of continuous geometry for objects like Cayley graphs of groups—which are fundamentally discrete but have infinite limits analogous to continuous systems. (Compare page 938.)

▪ **Manifold undecidability.** Given a particular set of network substitution rules there is in general no finite way to decide whether any sequence of such rules exists that will transform particular networks into each other. (Compare undecidability in multiway systems on page 779.) And although one might not expect it on the basis of traditional mathematical intuition, there is an analog of this even for topological equivalence of ordinary continuous manifolds. For the fundamental groups that represent how basic loops can be combined must be equivalent for equivalent manifolds. Yet it turns out that in 4D and above the fundamental group can have essentially any set of generators and relations—so that the undecidability of the word problem for arbitrary groups (see page 1141) implies undecidability of equivalence of manifolds. (In 2D it is straightforward to decide equivalence, and in 3D it is known that only some fundamental groups can be obtained—roughly because not all networks can be embedded in 2D—and it is expected that it will ultimately be possible to decide equivalence.)

▪ **Non-integer dimensions.** Unlike in traditional differential geometry (and general relativity) my formulation of space as a network potentially allows concepts like curvature to be defined even outside of integers numbers of dimensions.

▪ **Page 534 · Lorentzian spaces.** In ordinary Euclidean space distance is given by $Sqrt[x^2 + y^2 + z^2]$. In setting up relativity theory it is convenient (see page 1042) to define an analog of distance (so-called proper time) in 4D spacetime by $Sqrt[c^2 t^2 - x^2 - y^2 - z^2]$. And in terms of differential geometry such Minkowski space can be specified by the metric $DiagonalMatrix[\{+1, -1, -1, -1\}]$ (now taking $c = 1$). To set up general relativity one then considers not Riemannian manifolds but instead Lorentzian ones in which the metric is not positive definite, but instead has the signature of Minkowski space.

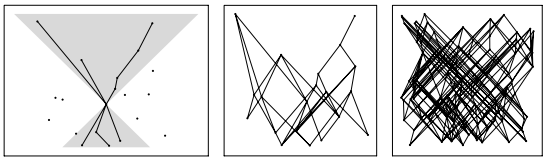
In such Lorentzian spaces, however, there is no useful immediate analog of a sphere. For given any point, even the light cone that corresponds to points at zero spacetime distance from it has an infinite volume. But with an appropriate definition one can still set up cones that have finite volume. To do this in general one starts by picking a vector e in a timelike direction, then normalizes it to be a unit vector so that $e \cdot g \cdot e == -1$. Then one defines a cone of height t whose apex is a given point to be those points whose displacement

vector v satisfies $0 > e \cdot g \cdot v > -t$ (and $0 > v \cdot g \cdot v$). And the volume of such a cone then turns out to be

$$s[d]\, t^{d+1} (1 - t^2 (d+1)(d\, \text{RicciScalar} + 2(d+1)(\text{RicciTensor} \cdot e \cdot e))/((d+2)(d+3)) + \ldots)/(d+1)$$

■ **Torsion.** In standard geometry, one assumes that the distance from one point to another is the same as the distance back, so that the metric tensor can be taken to be symmetric, and there is zero so-called torsion. But in for example a causal network, connections have definite directions, and there is in general no such symmetry. And if one looks at the volume of a cone this can then introduce a correction proportional to r. But as soon as there is enough uniformity to define a reasonable notion of static space, it seems that this effect must vanish. (Note that in pure mathematics there are several different uses of the word "torsion". Here I use it to refer to the antisymmetric parts of the metric tensor.)

■ **Random causal networks.** If one assumes that there are events at random positions in continuous spacetime, then one can construct an effective causal network for them by setting up connections between each event and all events in its future light cone—then deleting connections that are redundant in the sense that they just provide shortcuts to events that could otherwise be reached by following multiple connections. The pictures below show examples of causal networks obtained in this way. The number of connections generally increases faster than linearly with the number of events. Most links end up being at angles that are close to the edge of the light cone.



■ **Page 534 · Einstein equations.** In the absence of matter, the standard statement of the Einstein equations is that all components of the Ricci tensor—and thus also the Ricci scalar—must be zero (or formally that $R_{ij} = 0$). But since the vanishing of all components of a tensor must be independent of the coordinates used, it follows that the vacuum Einstein equations are equivalent to the statement $\text{RicciTensor} \cdot e \cdot e == 0$ for all timelike unit vectors e—a statement that can readily be applied to networks of the kind I consider in the main text. (A related statement is that the 3D Ricci scalar curvature of all spacelike hypersurfaces must vanish wherever these have vanishing extrinsic curvature.)

Another way to state the Einstein equations—already discussed by David Hilbert in 1915—is as the constraint that the integral of $\text{RicciScalar}\ \text{Sqrt}[\text{Det}[g]]$ (the so-called Einstein-Hilbert action) be an extremum. (An idealized soap film or other minimal surface extremizes the integral of the intrinsic volume element $\text{Sqrt}[\text{Det}[g]]$, without a RicciScalar factor.) In the discrete Regge calculus that I mention on page 1054 this variational principle turns out to have a rather simple form.

The Einstein-Hilbert action—and the Einstein equations—can be viewed as having the simplest forms that do not ultimately depend on the choice of coordinates. Higher-order terms—say powers of the Ricci scalar curvature—could well arise from underlying network systems, but would not contribute noticeably except in very high gravitational fields.

Various physical interpretations can be given of the vanishing of the Ricci tensor implied by the ordinary vacuum Einstein equations. Closely related to my discussion of the absence of t^2 terms in volume growth for 4D spacetime cones is the statement that if one sets up a small 3D ball of comoving test particles then the volume it defines must have zero first and second derivatives with time.

Below 4D the vanishing of the Ricci tensor immediately implies the vanishing of all components of the Riemann tensor—so that the vacuum Einstein equations force space at least locally to have its ordinary flat form. (Even in 2D there can nevertheless still be non-trivial global topology—for example with flat space having its edges identified as on a torus. In the Euclidean case there were for a long time no non-trivial solutions to the Einstein equations known in any number of dimensions, but in the 1970s examples were found, including large families of Calabi-Yau manifolds.)

In the presence of matter, the typical formal statement of the full Einstein equations is $R_{\mu\nu} - R\, g_{\mu\nu}/2 = 8\pi\, G\, T_{\mu\nu}/c^4$, where $T_{\mu\nu}$ is the energy-momentum (stress-energy) tensor for matter and G is the gravitational constant. (An additional so-called cosmological term $\lambda\, g_{\mu\nu}$ is sometimes added on the right to adjust the effective overall energy density of the universe, and thus its expansion rate. Note that the equation can also be written $R_{\mu\nu} = 8\pi\, G (T_{\mu\nu} - 1/2\, T^{\mu}_{\mu}\, g_{\mu\nu})/c^4$.) The μ, ν component of $T_{\mu\nu}$ gives the flux of the μ component of 4-momentum (whose components are energy and ordinary 3-momentum) in the ν direction. The fact that T_{00} is energy density implies that for static matter (where $E = m c^2$) the equation is in a sense a minimal extension of Poisson's equation of Newtonian gravity theory. Note that conservation of energy and momentum implies that $T_{\mu\nu}$ must have zero divergence—a result guaranteed in the Einstein equations by the structure of the left-hand side.

In the variational approach to gravity mentioned above, the *RicciScalar* plays the role of a Lagrangian density for pure gravity—and in the presence of matter the Lagrangian density for matter must be added to it. At a physical level, the full Einstein equations can be interpreted as saying that the volume v of a small ball of comoving test particles satisfies

$$\partial_{tt} v[t]/v[t] == -1/2\,(\rho + 3\,p)$$

where ρ is the total energy density and p is the pressure averaged over all space directions.

To solve the full Einstein equations in any particular physical situation requires a knowledge of $T_{\mu\nu}$—and thus of properties of matter such as the relation between pressure and energy density (equation of state). Quite a few global results about the formation of singularities and the absence of paths looping around in time can nevertheless be obtained just by assuming certain so-called energy conditions for $T_{\mu\nu}$. (A fairly stringent example is $0 \le p \le \rho/3$—and whether this is actually true for non-trivial interacting quantum fields remains unclear.)

In their usual formulation, the Einstein equations are thought of as defining constraints on the structure of 4D spacetime. But at some level they can also be viewed as defining how 3D space evolves with time. And indeed the so-called initial value formulations constructed in the 1960s allow one to start with a 3D metric and various extrinsic curvatures defined for a 3D spacelike hypersurface, and then work out how these change on successive hypersurfaces. But at least in terms of tensors, the equations involved show nothing like the simplicity of the usual 4D Einstein equations. One can potentially view the causal networks that I discuss in the main text as providing another approach to setting up an initial value formulation of the Einstein equations.

▪ **Page 536 · Pure gravity.** In the absence of matter, the Einstein equations always admit ordinary flat Minkowski space as a solution. But they also admit other solutions that in effect represent configurations of pure gravitational field. And in fact the 4D vacuum Einstein equations are already a sophisticated set of nonlinear partial differential equations that can support all sorts of complex behavior. Several tens of families of solutions to the equations have been found—some with obvious physical interpretations, others without.

Already in 1916 Karl Schwarzschild gave the solution for a spherically symmetric gravitational field. He imagined that this field itself existed in a vacuum—but that it was produced by a mass such as a star at its center. In its original form the metric becomes singular at radius $2\,G\,m/c^2$ (or $3\,m$ km with m in solar masses). At first it was assumed that this would always be inside a star, where the vacuum Einstein equations would not apply. But in the 1930s it was suggested that stars could collapse to concentrate their mass in a smaller radius. The singularity was then interpreted as an event horizon that separates the interior of a black hole from the ordinary space around it. In 1960 it was realized, however, that appropriate coordinates allowed smooth continuation across the event horizon—and that the only genuine singularity was infinite curvature at a single point at the center. Sometimes it was said that this must reflect the presence of a point mass, but soon it was typically just said to be a point at which the Einstein equations—for whatever reason—do not apply. Different choices of coordinates led to different apparent locations and forms of the singularity, and by the late 1970s the most common representation was just a smooth manifold with a topology reflecting the removal of a point—and without any specific reference to the presence of matter.

Appealing to ideas of Ernst Mach from the late 1800s it has often been assumed that to get curvature in space always eventually requires the presence of matter. But in fact even the vacuum Einstein equations for complete universes (with no points left out) have solutions that show curvature. If one assumes that space is both homogeneous and isotropic then it turns out that only ordinary flat Minkowski space is allowed. (When matter or a cosmological term is present one gets different solutions—that always expand or contract, and are much studied in cosmology.) If anisotropy is present, however, then there can be all sorts of solutions—classified for example as having different Bianchi symmetry types. And a variety of inhomogeneous solutions with no singularities are also known—an example being the 1962 Ozsváth-Schücking rotating vacuum. But in all cases the structure is too simple to capture much that seems relevant for our present universe.

One form of solution to the vacuum Einstein equations is a gravitational wave consisting of a small perturbation propagating through flat space. No solutions have yet been found that represent complete universes containing emitters and absorbers of such waves (or even for example just two massive bodies). But it is known that combinations of gravitational waves can be set up that will for example evolve to generate singularities. And I suspect that nonlinear interactions between such waves will also inevitably lead to the analog of turbulence for pure gravity. (Numerical simulations often show all sorts of complex behavior—but in the past this has normally been attributed just to the approximations used. Note that for example Bianchi type IX solutions for a complete universe show sensitive dependence on initial conditions—and no doubt this can also happen with nonlinear gravitational waves.)

As mentioned on page 1028, Albert Einstein considered the possibility that particles of matter might somehow just be localized structures in gravitational and electromagnetic fields. And in the mid-1950s John Wheeler studied explicit simple examples of such so-called geons. But in all cases they were found to be unstable—decaying into ordinary gravitational waves. The idea of having purely gravitational localized structures has also occasionally been considered—but so far no stable field configuration has been found. (And no purely repetitive solutions can exist.)

The equivalence principle (see page 1047) might suggest that anything with mass—or energy—should affect the curvature of space in the same way. But in the Einstein equations the energy-momentum tensor is not supposed to include contributions from the gravitational field. (There are alternative and seemingly inelegant theories of gravity that work differently—and notably do not yield black holes. The setup is also somewhat different in recent versions of string theory.) The very definition of energy for the gravitational field is not particularly straightforward in general relativity. But perhaps a definition could be found that would allow localized structures in the gravitational field to make effective contributions to the energy-momentum tensor that would mimic those from explicit particles of matter. Nevertheless, there are quite a few phenomena associated with particles that seem difficult to reproduce with pure gravity—at least say without extra dimensions. One example is parity violation; another is the presence of long-range forces other than gravity.

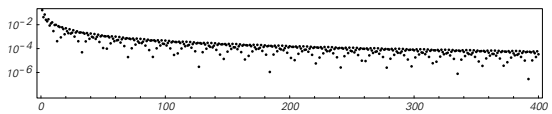
▪ **Quantum gravity.** That there should be quantum effects in gravity was already noted in the 1910s, and when quantum field theory began to develop in the 1930s, there were immediately attempts to apply it to gravity. The first idea was to represent gravity as a field that exists in flat spacetime, and by analogy with photons in quantum electrodynamics to introduce gravitons (at one point identified with neutrinos). By the mid-1950s a path integral (see page 1061) based on the Einstein-Hilbert action had been constructed, and by the early 1960s Feynman diagram rules had been derived, and it had been verified that tree diagrams involving gravitons gave results that agreed with general relativity for small gravitational fields. But as soon as loop diagrams were considered, infinities began to appear. And unlike for quantum electrodynamics there did not seem to be only a finite number of these—that could be removed by renormalization. And in fact by 1973 gravity coupled to matter had been shown for certain not to be renormalizable—and the same was finally shown for pure gravity in 1986. There was an attempt in the 1970s and early 1980s to look directly at the path integral—without doing an expansion in terms of Feynman diagrams. But despite the fact that at least in Euclidean spacetime a variety of seemingly relevant field configurations were identified, many mathematical difficulties were encountered. And in the late-1970s there began to be interest in the idea that supersymmetric field theories might make infinities associated with gravitons be cancelled by ones associated with other particles. But in the end this did not work out. And then in the mid-1980s one of the great attractions of string theory was that it seemed to support graviton excitations without the problem of infinities seen in point-particle field theories. But it had other problems, and to avoid these, supersymmetry had to be introduced, leading to the presence of many other particles that have so far not been observed. (See also page 1029.)

Starting in the 1950s a rather different approach to quantum gravity involved trying to find a representation of the structure of spacetime in which a quantum analog of the Einstein equations could be obtained by the formal procedure of canonical quantization (see page 1058). Yet despite a few signs of progress in the 1960s there was great difficulty in finding appropriately independent variables to use. In the late 1980s, however, it was suggested that variables could be used corresponding roughly to gravitational fluxes through loops in space. And in terms of these loop variables it was at least formally possible to write down a version of quantum gravity. Yet while this was found in the 1990s to have a correspondence with spin networks (see below), it has remained impossible to see just how it might yield ordinary general relativity as a limit.

Even if one assumes that spacetime is in a sense ultimately continuous one can imagine investigating quantum gravity by doing some kind of discrete approximation. And in 1961 Tullio Regge noted that for a simplicial complex (see page 1050) the Einstein-Hilbert action has a rather simple form in terms of angles between edges. Starting in the 1980s after the development of lattice gauge theories, simulations of random surfaces and higher-dimensional spaces set up in this way were done—often using so-called dynamic triangulation based on random sequences of generalized Alexander moves from page 1038. But there were difficulties with Lorentzian spaces, and when large-scale average behavior was studied, it seemed difficult to reproduce observed smooth spacetime. Analytical approaches (that happened to be like 0D string theory) were also found for 2D discrete spacetimes (compare page 1038)—but they were not successfully extended to higher dimensions.

Over the years, various attempts have been made to derive quantum gravity from fundamentally discrete models of

spacetime (compare page 1027). In recent times the most widely discussed have been spin networks—which despite their name ultimately seem to have fairly little to do with the systems I consider. Spin networks were introduced in 1964 by Roger Penrose as a way to set up an intrinsically quantum mechanical model of spacetime. A simple analog involves a 2D surface made out of triangles whose edges have integer lengths j_i. If one computes the product of $Exp[i(j_1 + j_2 - j_3)]$ for all triangles, then it turns out for example that this quantity is extremized exactly when the whole surface is flat. In 3D one imagines breaking space into tetrahedra whose edge lengths correspond to discrete quantum spin values. And in 1968 Tullio Regge and Giorgio Ponzano suggested—almost as an afterthought in technical work on $6j$ symbols—that the quantum probability amplitude for any form of space might perhaps be given by the product of $6j$ symbols for the spins on each tetrahedron. The $SixJSymbol[\{j_1, j_2, j_3\}, \{j_4, j_5, j_6\}]$ are slightly esoteric objects that correspond to recoupling coefficients for the 3D rotation group SO(3), and that arose in 1940s studies of combinations of three angular momenta in atomic physics—and were often represented graphically as networks. For large j_i they are approximated by $Cos[\theta + \pi/4]/Sqrt[12 \pi v]$, where v is the volume of the tetrahedron and θ is a deficit angle. And from this it turns out that limits of products of $6j$ symbols correspond essentially to $Exp[i s]$, where s is the discrete form of the Einstein-Hilbert action—extremized by flat 3D space. (The picture below shows for example $Abs[SixJSymbol[\{j, j, j\}, \{j, j, j\}]]$. Note that for any j the $6j$ symbols can be given in terms of *HypergeometricPFQ*.)



In the early 1990s there was again interest in spin networks when the Turaev-Viro invariant for 3D spaces was discovered from a topological field theory involving triangulations weighted with $6j$ symbols of the quantum group $SU(2)_q$—and it was seen that invariance under Alexander moves on the triangulation corresponded to the Biedenharn-Elliott identity for $6j$ symbols. In the mid-1990s it was then found that states in 3D loop quantum gravity (see above) could be represented in terms of spin networks—leading for example to quantization of all areas and volumes. In attempting extensions to 4D, spin foams have been introduced—and variously interpreted in terms of simplified Feynman diagrams, constructs in multidimensional category theory, and possible evolutions of spin networks. In all cases, however, spin networks and spin foams seem to be viewed just as calculational constructs that must be evaluated and added together to get quantum amplitudes—quite different from my idea of associating an explicit evolution history for the universe with the evolution of a network.

▪ **Cosmology.** On a large scale our universe appears to show a uniform expansion that makes progressively more distant galaxies recede from us at progressively higher speeds. In general relativity this is explained by saying that the initial conditions must have involved expansion—and that there is not enough in the way of matter or gravitational fields to produce the gravity to slow down this expansion too much. (Note that as soon as objects get gravitationally bound—like galaxies in clusters—there is no longer expansion between them.) The standard big bang model assumes that the universe starts with matter at what is in effect an arbitrarily high temperature. One issue much discussed in cosmology since the late 1970s is how the universe manages to be so uniform. Thermal equilibrium should eventually lead to uniformity—but different parts of the universe cannot come to equilibrium until there has at least been time for effects to propagate between them. Yet there seems for example to be overall uniformity in what we see if we look in opposite directions in the sky—even though extrapolating from the current rate of expansion there has not been enough time since the beginning of the universe for anything to propagate from one side to the other. But starting in the early 1980s it has been popular to think that early in its history the universe must have undergone a period of exponential expansion or so-called inflation. And what this would do is to take just a tiny region and make it large enough to correspond to everything we can now see in the universe. But the point is that a sufficiently tiny region will have had time to come to thermal equilibrium—and so will be approximately uniform, just as the cosmic microwave background is now observed to be. The actual process of inflation is usually assumed to reflect some form of phase transition associated with decreasing temperature of matter in the universe. Most often it is assumed that in the present universe a delicate balance must exist between energy density from a background Higgs field (see page 1047) and a cosmological term in the Einstein equations (see page 1052). But above a critical temperature thermal fluctuations should prevent the background from forming—leading to at least some period in which the universe is dominated by a cosmological term which yields exponential expansion. There tend to be various detailed problems with this scenario, but at least with a sufficiently complicated setup it seems possible to get results that are consistent with observations made so far.

In the context of the discoveries in this book, my expectation is that the universe started from a simple small network, then

progressively added more and more nodes as it evolved, until eventually on a large scale something corresponding to 4D spacetime emerged. And with this setup, the observed uniformity of the universe becomes much less surprising. Intrinsic randomness generation always tends to lead to a certain uniformity in networks. But the crucial point is that this will not take long to happen throughout any network if it is appropriately connected. Traditional models tend to assume that there are ultimately a fixed number of spacetime dimensions in the universe. And with this assumption it is inevitable that if the universe in a sense expands at the speed of light, then regions on opposite sides of it can essentially never share any common history. But in a network model the situation is different. The causal network always captures what happens. And in a case like page 518—with spacetime always effectively having a fixed finite dimension—points that are a distance t apart tend to have common ancestors only at least t steps back. But in a case like (a) on page 514—where spacetime has the structure of an exponentially growing tree—points a distance t apart typically have common ancestors just *Log[t]* steps back. And in fact many kinds of causal networks—say associated with early randomly connected space networks—will inevitably yield common ancestors for distant parts of the universe. (Note that such phenomena presumably occur at around the Planck scale of 10^{19} GeV rather than at the 10^{15} GeV or lower scale normally discussed in connection with inflation. They can to some extent be captured in general relativity by imagining an effective spacetime dimension that is initially infinite, then gradually decreases to 4.)

Quantum Phenomena

■ **History.** In classical physics quantities like energy were always assumed to correspond to continuous variables. But in 1900 Max Planck noticed that fits to the measured spectrum of electromagnetic radiation produced by hot objects could be explained if there were discrete quanta of electromagnetic energy. And by 1910 work by Albert Einstein, notably on the photoelectric effect and on heat capacities of solids, had given evidence for discrete quanta of energy in both light and matter. In 1913 Niels Bohr then made the suggestion that the discrete spectrum of light emitted by hydrogen atoms could be explained as being produced by electrons making transitions between orbits with discrete quantized angular momenta. By 1920 ideas from celestial mechanics had been used to develop a formalism for quantized orbits which successfully explained various features of atoms and chemical elements. But it was not clear how to extend the formalism say to a problem like propagation of light through a crystal. In 1925, however, Werner Heisenberg suggested a new and more general formalism that became known as matrix mechanics. The original idea was to imagine describing the state of an atom in terms of an array of amplitudes for virtual oscillators with each possible frequency. Particular conditions amounting to quantization were then imposed on matrices of transitions between these, and the idea was introduced that only certain kinds of amplitude combinations could ever be observed. In 1923 Louis de Broglie had suggested that just as light—which in optics was traditionally described in terms of waves—seemed in some respects to act like discrete particles, so conversely particles like electrons might in some respects act like waves. In 1926 Erwin Schrödinger then suggested a partial differential equation for the wave functions of particles like electrons. And when effectively restricted to a finite region, this equation allowed only certain modes, corresponding to discrete quantum states—whose properties turned out to be exactly the same as implied by matrix mechanics. In the late 1920s Paul Dirac developed a more abstract operator-based formalism. And by the end of the 1920s basic practical quantum mechanics was established in more or less the form it appears in textbooks today. In the period since, increasing computational capabilities have allowed coupled Schrödinger equations for progressively more particles to be solved (reasonably accurate solutions for hundreds of particles can now be found), allowing ever larger studies in atomic, molecular, nuclear and solid-state physics. A notable theoretical interest starting in the 1980s was so-called quantum chaos, in which it was found that modes (wave functions) in regions like stadiums that did not yield simple analytical solutions tended to show complicated and seemingly random forms.

Basic quantum mechanics is set up to describe how fixed numbers of particles behave—say in externally applied electromagnetic or other fields. But to describe things like fields one must allow particles to be created and destroyed. In the mid-1920s there was already discussion of how to set up a formalism for this, with an underlying idea again being to think in terms of virtual oscillators—but now one for each possible state of each possible one of any number of particles. At first this was just applied to a pure electromagnetic field of non-interacting photons, but by the end of the 1920s there was a version of quantum electrodynamics (QED) for interacting photons and electrons that is essentially the same as today. To find predictions from this theory a so-called perturbation expansion was made, with successive terms representing progressively more interactions, and each

having a higher power of the so-called coupling constant $\alpha \simeq 1/137$. It was immediately noticed, however, that self-interactions of particles would give rise to infinities, much as in classical electromagnetism. At first attempts were made to avoid this by modifying the basic theory (see page 1044). But by the mid-1940s detailed calculations were being done in which infinite parts were just being dropped—and the results were being found to agree rather precisely with experiments. In the late 1940s this procedure was then essentially justified by the idea of renormalization: that since in all possible QED processes only three different infinities can ever appear, these can in effect systematically be factored out from all predictions of the theory. Then in 1949 Feynman diagrams were introduced (see note below) to represent terms in the QED perturbation expansion—and the rules for these rapidly became what defined QED in essentially all practical applications. Evaluating Feynman diagrams involved extensive algebra, and indeed stimulated the development of computer algebra (including my own interest in the field). But by the 1970s the dozen or so standard processes discussed in QED had been calculated to order α^2—and by the mid-1980s the anomalous magnetic moment of the electron had been calculated to order α^4, and nearly one part in a trillion (see note below).

But despite the success of perturbation theory in QED it did not at first seem applicable to other issues in particle physics. The weak interactions involved in radioactive beta decay seemed too weak for anything beyond lowest order to be relevant—and in any case not renormalizable. And the strong interactions responsible for holding nuclei together (and associated for example with exchange of pions and other mesons) seemed too strong for it to make sense to do an expansion with larger numbers of individual interactions treated as less important. So this led in the 1960s to attempts to base theories just on setting up simple mathematical constraints on the overall so-called S matrix defining the mapping from incoming to outgoing quantum states. But by the end of the 1960s theoretical progress seemed blocked by basic questions about functions of several complex variables, and predictions that were made did not seem to work well.

By the early 1970s, however, there was increasing interest in so-called gauge or Yang-Mills theories formed in essence by generalizing QED to operate not just with a scalar charge, but with charges viewed as elements of non-Abelian groups. In 1972 it was shown that spontaneously broken gauge theories of the kind needed to describe weak interactions were renormalizable—allowing meaningful use of perturbation theory and Feynman diagrams. And then in 1973 it was discovered that QCD—the gauge theory for quarks and gluons with SU(3) color charges—was asymptotically free (it was known to be renormalizable), so that for processes probing sufficiently small distances, its effective coupling was small enough for perturbation theory. By the early 1980s first-order calculations of most basic QCD processes had been done—and by the 1990s second-order corrections were also known. Schemes for adding up all Feynman diagrams with certain very simple repetitive or other structures were developed. But despite a few results about large-distance analogs of renormalizability, the question of what QCD might imply for processes at larger distances could not really be addressed by such methods.

In 1941 Richard Feynman pointed out that amplitudes in quantum theory could be worked out by using path integrals that sum with appropriate weights contributions from all possible histories of a system. (The Schrödinger equation is like a diffusion equation in imaginary time, so the path integral for it can be thought of as like an enumeration of random walks. The idea of describing random walks with path integrals was discussed from the early 1900s.) At first the path integral was viewed mostly as a curiosity, but by the late 1970s it was emerging as the standard way to define a quantum field theory. Attempts were made to see if the path integral for QCD (and later for quantum gravity) could be approximated with a few exact solutions (such as instantons) to classical field equations. By the early 1980s there was then extensive work on lattice gauge theories in which the path integral (in Euclidean space) was approximated by randomly sampling discretized field configurations. But—I suspect for reasons that I discuss in the note below—such methods were never extremely successful. And the result is that beyond perturbation theory there is still no real example of a definitive success from standard relativistic quantum field theory. (In addition, even efforts in the context of so-called axiomatic field theory to set up mathematically rigorous formulations have run into many difficulties—with the only examples satisfying all proposed axioms typically in the end being field theories without any real interactions. In condensed matter physics there are nevertheless cases like the Kondo model where exact solutions have been found, and where the effective energy function for electrons happens to be roughly the same as in a relativistic theory.)

As mentioned on page 1044, ordinary quantum field theory in effect deals only with point particles. And indeed a recurring issue in it has been difficulty with constraints and redundant degrees of freedom—such as those associated with extended objects. (A typical goal is to find variables in which one can carry out what is known as canonical quantization: essentially applying the same straightforward

transformation of equations that happens to work in ordinary elementary quantum mechanics.) One feature of string theory and its generalizations is that they define presumably consistent quantum field theories for excitations of extended objects—though an analog of quantum field theory in which whole strings can be created and destroyed has not yet been developed.

When the formalism of quantum mechanics was developed in the mid-1920s there were immediately questions about its interpretation. But it was quickly suggested that given a wave function ψ from the Schrödinger equation $Abs[\psi]^2$ should represent probability—and essentially all practical applications have been based on this ever since. From a conceptual point of view it has however often seemed peculiar that a supposedly fundamental theory should talk only about probabilities. Following the introduction of the uncertainty principle and related formalism in the 1920s one idea that arose was that—in rough analogy to relativity theory—it might just be that there are only certain quantities that are observable in definite ways. But this was not enough, and by the 1930s it was being suggested that the validity of quantum mechanics might be a sign that whole new general frameworks for philosophy or logic were needed—a notion supported by the apparent need to bring consciousness into discussions about measurement in quantum mechanics (see page 1063). The peculiar character of quantum mechanics was again emphasized by the idealized experiment of Albert Einstein, Boris Podolsky and Nathan Rosen in 1935. But among most physicists the apparent lack of an ordinary mechanistic way to think about quantum mechanics ended up just being seen as another piece of evidence for the fundamental role of mathematical formalism in physics.

One way for probabilities to appear even in deterministic systems is for there to be hidden variables whose values are unknown. But following mathematical work in the early 1930s it was usually assumed that this could not be what was going on in quantum mechanics. In 1952 David Bohm did however manage to construct a somewhat elaborate model based on hidden variables that gave the same results as ordinary quantum mechanics—though involved infinitely fast propagation of information. In the early 1960s John Bell then showed that in any hidden variables theory of a certain general type there are specific inequalities that combinations of probabilities must satisfy (see page 1064). And by the early 1980s experiments had shown that such inequalities were indeed violated in practice—so that there were in fact correlations of the kind suggested by quantum mechanics. At first these just seemed like isolated esoteric effects, but by the mid-1990s they were being codified in the field of quantum information theory, and led to constructions with names like quantum cryptography and quantum teleportation.

Particularly when viewed in terms of path integrals the standard formalism of quantum theory tends to suggest that quantum systems somehow do more computation in their evolution than classical ones. And after occasional discussion as early as the 1950s, this led by the late 1980s to extensive investigation of systems that could be viewed as quantum analogs of idealized computers. In the mid-1990s efficient procedures for integer factoring and a few other problems were suggested for such systems, and by the late 1990s small experiments on these were beginning to be done in various types of physical systems. But it is becoming increasingly unclear just how the idealizations in the underlying model really work, and to what extent quantum mechanics is actually in the end even required—as opposed, say, just to classical wave phenomena. (See page 1147.)

Partly as a result of discussions about measurement there began to be questions in the 1980s about whether ordinary quantum mechanics can describe systems containing very large numbers of particles. Experiments in the 1980s and 1990s on such phenomena as macroscopic superposition and Bose-Einstein condensation nevertheless showed that standard quantum effects still occur with trillions of atoms. But inevitably the kinds of general phenomena that I discuss in this book will also occur—leading to all sorts of behavior that at least cannot readily be foreseen just from the basic rules of quantum mechanics.

▪ **Quantum effects.** Over the years, many suggested effects have been thought to be characteristic of quantum systems:

- Basic quantization (1913): mechanical properties of particles in effectively bounded systems are discrete;
- Wave-particle duality (1923): objects like electrons and photons can be described as either waves or particles;
- Spin (1925): particles can have intrinsic angular momentum even if they are of zero size;
- Non-commuting measurements (1926): one can get different results doing measurements in different orders;
- Complex amplitudes (1926): processes are described by complex probability amplitudes;
- Probabilism (1926): outcomes are random, though probabilities for them can be computed;
- Amplitude superposition (1926): there is a linear superposition principle for probability amplitudes;
- State superposition (1926): quantum systems can occur in superpositions of measurable states;

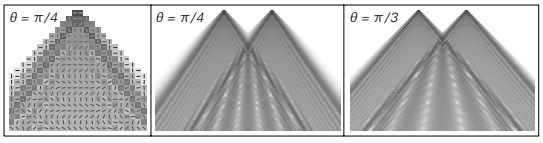
- Exclusion principle (1926): amplitudes cancel for fermions like electrons to go in the same state;
- Interference (1927): probability amplitudes for particles can interfere, potentially destructively;
- Uncertainty principle (1927): quantities like position and momenta have related measurement uncertainties;
- Hilbert space (1927): states of systems are represented by vectors of amplitudes rather than individual variables;
- Field quantization (1927): only discrete numbers of any particular kind of particle can in effect ever exist;
- Quantum tunnelling (1928): particles have amplitudes to go where no classical motion would take them;
- Virtual particles (1932): particles can occur for short times without their usual energy-momentum relation;
- Spinors (1930s): fermions show rotational invariance under SU(2) rather than SO(3);
- Entanglement (1935): separated parts of a system often inevitably behave in irreducibly correlated ways;
- Quantum logic (1936): relations between events do not follow ordinary laws of logic;
- Path integrals (1941): probabilities for behavior are obtained by summing contributions from many paths;
- Imaginary time (1947): statistical mechanics is like quantum mechanics in imaginary time;
- Vacuum fluctuations (1948): there are continual random field fluctuations even in the vacuum;
- Aharanov-Bohm effect (1959): magnetic fields can affect particles even in regions where they have zero strength;
- Bell's inequalities (1964): correlations between events can be larger than in any ordinary probabilistic system;
- Anomalies (1969): virtual particles can have effects that violate the original symmetries of a system;
- Delayed choice experiments (1978): whether particle or wave features are seen can be determined after an event;
- Quantum computing (1980s): there is the potential for fundamental parallelism in computations.

All of these effects are implied by the standard mathematical formalism of quantum theory. But it has never been entirely clear which of them are in a sense true defining features of quantum phenomena, and which are somehow just details. It does not help that most of the effects—at least individually—can be reproduced by mechanisms that seem to have little to do with the usual structure of quantum theory. So for example there will tend to be quantization whenever the underlying elements of a system are discrete. Similarly, features like the uncertainty principle and path integrals tend to be seen whenever things like waves are involved. And probabilistic effects can arise from any of the mechanisms for randomness discussed in Chapter 7. Complex amplitudes can be thought of just as vector quantities. And it is straightforward to set up rules that will for example reproduce the detailed evolution of amplitudes according say to the Schrödinger equation (see note below). It is somewhat more difficult to set up a system in which such amplitudes will somehow directly determine probabilities. And indeed in recent times consequences of this—such as violations of Bell's inequalities—are what have probably most often been quoted as the most unique features of quantum systems. It is however notable that the vast majority of traditional applications of quantum theory do not seem to have anything to do with such effects. And in fact I do not consider it at all clear just what is really essential about them, and what is in the end just a consequence of the extreme limits that seem to need to be taken to get explicit versions of them.

▪ **Reproducing quantum phenomena.** Given molecular dynamics it is much easier to see how to reproduce fluid mechanics than rigid-body mechanics—since to get rigid bodies with only a few degrees of freedom requires taking all sorts of limits of correlations between underlying molecules. And I strongly suspect that given a discrete underlying model of the type I discuss here it will similarly be much easier to reproduce quantum field theory than ordinary quantum mechanics. And indeed even with traditional formalism, it is usually difficult to see how quantum mechanics can be obtained as a limit of quantum field theory. (Classical limits are slightly easier: they tend to be associated with stationary features or caustics that occur at large quantum numbers—or coherent states that represent eigenstates of raising or particle creation operators. Note that the exclusion principle makes classical limits for fermions difficult—but crucial for the stability of bulk matter.)

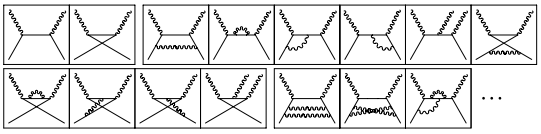
▪ **Discrete quantum mechanics.** While there are many issues in finding a complete underlying discrete model for quantum phenomena, it is quite straightforward to set up continuous cellular automata whose limiting behavior reproduces the evolution of probability amplitudes in standard quantum mechanics. One starts by assigning a continuous complex number value to each cell. Then given the list of such values the crucial constraint imposed by the standard formalism of quantum mechanics is unitarity: that the quantity $Tr[Abs[list]^2]$ representing total probability should be conserved. This is in a sense analogous to conservation of total density in diffusion processes. From

the discussion of page 1024 one can reproduce the 1D diffusion equation with a continuous block cellular automaton in which the new value of each block is given by $\{\{1-\xi, \xi\}, \{\xi, 1-\xi\}\} . \{a_1, a_2\}$. So in the case of quantum mechanics one can consider having each new block be given by $\{\{\text{Cos}[\theta], i\,\text{Sin}[\theta]\}, \{i\,\text{Sin}[\theta], \text{Cos}[\theta]\}\} . \{a_1, a_2\}$. The pictures below show examples of behavior obtained with this rule. (Gray levels represent magnitude for each cell, and arrows phase.) And it turns out that in suitable limits one generally gets essentially the behavior expected from either the Dirac or Klein-Gordon equations for relativistic particles, or the Schrödinger equation for non-relativistic particles. (Versions of this were noticed by Richard Feynman in the 1940s in connection with his development of path integrals, and were pointed out again several times in the 1980s and 1990s.)



One might hope to be able to get an ordinary cellular automaton with a limited set of possible values by choosing a suitable θ. But in fact in non-trivial cases most of the cells generated at each step end up having distinct values. One can generalize the setup to more dimensions or to allow $n \times n$ matrices that are elements of SU(n). Such matrices can be viewed in the context of ordinary quantum formalism as S matrices for elementary evolution events—and can in general represent interactions. (Note that all rules based on matrices are additive, reflecting the usual assumption of linearity at the level of amplitudes in quantum mechanics. Non-additive unitary rules can also be found. The analog of an external potential can be introduced by progressively changing values of certain cells at each step. Despite their basic setup the systems discussed here are not direct analogs of standard quantum spin systems, since these normally have local Hamiltonians and non-local evolution functions, while the systems here have local evolution functions but seem always to require non-local Hamiltonians.)

▪ **Page 540 · Feynman diagrams.** The pictures below show a typical set of Feynman diagrams used to do calculations in QED—in this case for so-called Compton scattering of a photon by an electron. The straight lines in the diagrams represent electrons; the wavy ones photons. At some level each diagram can be thought of as representing a process in which an electron and photon come in from the left, interact in some way, then go out to the right. The incoming and outgoing lines correspond to real particles that propagate to infinity. The lines inside each diagram correspond to virtual particles that in effect propagate only a limited distance, and have a distribution of energy-momentum and polarization properties that can differ from real particles. (Exchanges of virtual photons can be thought of as producing familiar electromagnetic forces; exchanges of virtual electrons as yielding an analog of covalent forces in chemistry.)



To work out the total probability for a process from Feynman diagrams, what one does is to find the expression corresponding to each diagram, then one adds these up, and squares the result. The first two blocks of pictures above show all the diagrams for Compton scattering that involve 2 or 3 photons—and contribute through order α^3. Since for QED $\alpha \simeq 1/137$, one might expect that this would give quite an accurate result—and indeed experiments suggest that it does. But the number of diagrams grows rapidly with order, and in fact the k^{th} order term can be about $(-1)^k \alpha^k (k/2)!$, yielding a series that formally diverges. In simpler examples where exact results are known, however, the first few terms typically still seem to give numerically accurate results for small α. (The high-order terms often seem to be associated with asymptotic series for things like $\text{Exp}[-1/\alpha]$.)

The most extensive calculation made so far in QED is for the magnetic moment of the electron. Ignoring parts that depend on particle masses the result (derived in successive orders from 1, 1, 7, 72, 891 diagrams) is

$$2\,(1 + \alpha/(2\,\pi) + (3\,\text{Zeta}[3]/4 - 1/2\,\pi^2\,\text{Log}[2] + \pi^2/12 + 197/144)\,(\alpha/\pi)^2 + (83/72\,\pi^2\,\text{Zeta}[3] - 215\,\text{Zeta}[5]/24 - 239\,\pi^4/2160 + 139\,\text{Zeta}[3]/18 + 25/18\,(24\,\text{PolyLog}[4, 1/2] + \text{Log}[2]^4 - \pi^2\,\text{Log}[2]^2) - 298/9\,\pi^2\,\text{Log}[2] + 17101\,\pi^2/810 + 28259/5184)\,(\alpha/\pi)^3 - 1.4\,(\alpha/\pi)^4 + \ldots)$$

or roughly

$$2. + 0.32\,\alpha - 0.067\,\alpha^2 + 0.076\,\alpha^3 - 0.029\,\alpha^4 + \ldots$$

The comparative simplicity of the symbolic forms here (which might get still simpler in terms of suitable generalized polylogarithm functions) may be a hint that methods much more efficient than explicit Feynman diagram evaluation could be used. But it seems likely that there would be limits to this, and that in the end QED will exhibit the kind of computational irreducibility that I discuss in Chapter 12.

Feynman diagrams in QCD work at the formal level very much like those in QED—except that there are usually many more of them, and their numerical results tend to be larger,

with expansion parameters often effectively being $\alpha\,\pi$ rather than α/π. For processes with large characteristic momentum transfers in which the effective α in QCD is small, remarkably accurate results are obtained with first and perhaps second-order Feynman diagrams. But as soon as the effective α becomes larger, Feynman diagrams as such rapidly seem to stop being useful.

■ **Quantum field theory.** In standard approaches to quantum field theory one tends to think of particles as some kind of small perturbations in a field. Normally for calculations these perturbations are on their own taken to be plane waves of definite frequency, and indeed in many ways they are direct analogs of waves in classical field theories like those of electromagnetism or fluid mechanics. To investigate collisions between particles, one thus looks at what happens with multiple waves. In a system described by linear equations, there is always a simple superposition principle, and waves just pass through each other unchanged. But what in effect leads to non-trivial interactions between particles is the presence of nonlinearities. If these are small enough then it makes sense to do a perturbation expansion in which one approximates field configurations in terms of a succession of arrangements of ordinary waves—as in Feynman diagrams. But just as one cannot expect to capture fully turbulent fluid flow in terms of a few simple waves, so in general as soon as there is substantial nonlinearity it will no longer be sufficient just to do perturbation expansions. And indeed for example in QCD there are presumably many cases in which it is necessary to look at something closer to actual complete field configurations—and correlations in them.

The way the path integral for a quantum field theory works, each possible configuration of the field is in effect taken to make a contribution $Exp[i\,s/\hbar]$, where s is the so-called action for the field configuration (given by the integral of the Lagrangian density—essentially a modified energy density), and $\hbar$ is a basic scale factor for quantum effects (Planck's constant divided by $2\,\pi$). In most places in the space of all possible field configurations, the value of s will vary quite quickly between nearby configurations. And assuming this variation is somehow random, the contributions of these nearby configurations will tend to cancel out. But inevitably there will be some places in the space where s is stationary (has zero variational derivative) with respect to changes in fields. And in some approximation the field configurations in these places can be expected to dominate the path integral. But it turns out that these field configurations are exactly the ones that satisfy the partial differential equations for the classical version of the field theory. (This is analogous to what happens for example in classical diffraction theory, where there is an analog of the path integral—with $\hbar$ replaced by inverse frequency—whose stationary points correspond through the so-called eikonal approximation to rays in geometrical optics.) In cases like QED and QCD the most obvious solutions to the classical equations are ones in which all fields are zero. And indeed standard perturbation theory is based on starting from these and then looking at the expansion of $Exp[i\,s/\hbar]$ in powers of the coupling constant. But while this works for QED, it is only adequate for QCD in situations where the effective coupling is small. And indeed in other situations it seems likely that there will be all sorts of other solutions to the classical equations that become important. But apart from a few special cases with high symmetry, remarkably little is known about solutions to the classical equations even for pure gluon fields. No doubt the analog of turbulence can occur, and certainly there is sensitive dependence on initial conditions (even non-Abelian plane waves involve iterated maps that show this). Presumably much like in fluids there are various coherent structures such as color flux tubes and glueballs. But I doubt that states involving organized arrangements of these are common. And in general when there is strong coupling the path integral will potentially be dominated by large numbers of configurations not close to classical solutions.

In studying quantum field theories it has been common to consider effectively replacing time coordinates t by $i\,t$ to go from ordinary Minkowski space to Euclidean space (see page 1043). But while there is no problem in doing this at a formal mathematical level—and indeed the expressions one gets from Feynman diagrams can always be analytically continued in this way—what general correspondence there is for actual physical processes is far from clear. Formally continuing to Euclidean space makes path integrals easier to define with traditional mathematics, and gives them weights of the form $Exp[-\beta\,s]$—analogous to constant temperature systems in statistical mechanics. Discretizing yields lattice gauge theories with energy functions involving for example $Cos[\theta_i - \theta_j]$ for color directions at adjacent sites. And Monte Carlo studies of such theories suggest all sorts of complex behavior, often similar in outline from what appears to occur in the corresponding classical field theories. (It seems conceivable that asymptotic freedom could lead to an analog of damping at small scales roughly like viscosity in turbulent fluids.)

One of the apparent implications of QCD is the confinement of quarks and gluons inside color-neutral hadrons. And at some level this is presumably a reflection of the fact that QCD forces get stronger rather than weaker with increasing distance. The beginnings of this are visible in perturbation

theory in the increase of the effective coupling with distance associated with asymptotic freedom. (In QED effective couplings decrease slightly with distance because fields get screened by virtual electron-positron pairs. The same happens with virtual quarks in QCD, but a larger effect is virtual gluon pairs whose color magnetic moments line up with a color field and serve to increase it.) At larger distances something like color flux tubes that act like elastic strings may form. But no detailed way to get confinement with purely classical gluon fields is known. In the quantum case, a sign of confinement would be exponential decrease with spacetime area of the average phase of color flux through so-called Wilson loops—and this is achieved if there is in a sense maximal randomness in field configurations. (Note that it is not inconceivable that the formal problem of whether quarks and gluons can ever escape to infinity starting from some given class of field configurations may in general be undecidable.)

■ **Vacuum fluctuations.** As an analog of the uncertainty principle, one of the implications of the basic formalism of quantum theory is that an ordinary quantum field can in a sense never maintain precisely zero value, but must always show certain fluctuations—even in what one considers the vacuum. And in terms of Feynman diagrams the way this happens is by virtual particle-antiparticle pairs of all types and all energy-momenta continually forming and annihilating at all points in the vacuum. Insofar as such vacuum fluctuations are always exactly the same, however, they presumably cannot be detected. (In the formalism of quantum field theory, they are usually removed by so-called normal ordering. But without this every mode of any quantum system will show a zero-point energy $\hbar\omega/2$—positive in sign for bosons and negative for fermions, cancelling for perfect supersymmetry. Quite what gravitational effects such zero-point energy might have has never been clear.) If one somehow changes the space in which a vacuum exists, there can be directly observable effects of vacuum fluctuations. An example is the 1948 Casimir effect—in which the absence of low-energy (long wavelength) virtual particle pairs in the space between two metal plates (but not in the infinite space outside) leads to a small but measurable force of attraction between them. The different detailed patterns of modes of different fields in different spaces can lead to very different effective vacuum energies—often negative. And at least with the idealization of impermeable classical conducting boundaries one predicts (based on work of mine from 1981) the peculiar effect that closed cycles can be set up that systematically extract energy from vacuum fluctuations in a photon field.

If one has moving boundaries it turns out that vacuum fluctuations can in effect be viewed as producing real particles. And as known since the 1960s, the same is true for expanding universes. What happens in essence is that the modes of fields in different background spacetime structures differ to the point where zero-point excitations seem like actual particle excitations to a detector or observer calibrated to fields in ordinary fixed flat infinite spacetime. And in fact just uniform acceleration turns out to make detectors register real particles in a vacuum—in this case with a thermal spectrum at a temperature proportional to the acceleration. (Uniform rotation also leads to real particles, but apparently with a different spectrum.) As expected from the equivalence principle, a uniform gravitational field should produce the same effect. (Uniform electric fields lead in a formally similar way to production of charged particles.) And as pointed out by Stephen Hawking in 1974, black holes should also generate thermal radiation (at a temperature $\hbar c^3/(8\pi G k M)$). A common interpretation is that the radiated particles are somehow ones left behind when the other particle in a virtual pair goes inside the event horizon. (A similar explanation can be given for uniform acceleration—for which there is also an event horizon.) There has been much discussion of the idea that Hawking radiation somehow shows pure quantum states spontaneously turning into mixed ones, more or less as in quantum measurements. But presumably this is just a reflection of the idealization involved in looking at quantum fields in a fixed background classical spacetime. And indeed work in string theory in the mid-1990s may suggest ways in which quantum gravity configurations of black hole surfaces could maintain the information needed for the whole system to act as a pure state.

■ **Page 542 · Quantum measurement.** The basic mathematical formalism used in standard quantum theory to describe pure quantum processes deals just with vectors of probability amplitudes. Yet our everyday experience of the physical world is that we observe definite things to happen. And the way this is normally captured is by saying that when an observation is made the vector of amplitudes is somehow replaced by its projection s into a subspace corresponding to the outcome seen—with the probability of getting the outcome being taken to be determined by $s \cdot \text{Conjugate}[s]$.

At the level of pure quantum processes, the standard rules of quantum theory say that amplitudes should be added as complex numbers—with the result that they can for example potentially cancel each other, and generally lead to wave-like interference phenomena. But after an observation is made, it is in effect assumed that a system can be described by ordinary real-number probabilities—so that for example no

interference is possible. (At a formal level, results of pure quantum processes are termed pure quantum states, and are characterized by vectors of probability amplitudes; results of all possible observations are termed mixed states, and are in effect represented as mixtures of pure states.)

Ever since the 1930s there have been questions about just what should count as an observation. To explain everyday experience, conscious perception presumably always must. But it was not clear whether the operation of inanimate measuring devices of various kinds also should. And a major apparent problem was that if everything—including the measuring device—is supposed to be treated as part of the same quantum system, then all of it must follow the rules for pure quantum processes, which do not explicitly include any reduction of the kind supposed to occur in observations.

One approach to getting around this suggested in the late 1950s is the many-worlds interpretation (see page 1035): that there is in a sense a universal pure quantum process that involves all possible outcomes for every conceivable observation, and that represents the tree of all possible threads of history—but that in a particular thread, involving a particular sequence of tree branches, and representing a particular thread of experience for us, there is in effect a reduction in the pure quantum process at each branch point. Similar schemes have been popular in quantum cosmology since the early 1990s in connection with studying wave functions for the complete universe.

A quite different—and I think much more fruitful—approach is to consider analyzing actual potential measurement processes in the context of ordinary quantum mechanics. For even if one takes these processes to be pure quantum ones, what I believe is that in almost all cases appropriate idealized limits of them will reproduce what are in effect the usual rules for observations in quantum theory. A key point is that for one to consider something a reasonable measurement it must in a sense yield a definitive result. And in the context of standard quantum theory this means that somehow all the probability amplitudes associated with the measuring device must in effect be concentrated in specific outcomes—with no significant interference between different outcomes.

If one has just a few quantum particles—governed say by an appropriate Schrödinger equation—then presumably there can be no such concentration. But with a sufficiently large number of particles—and appropriate interactions—one expects that there can be. At first this might seem impossible. For the basic rules for pure quantum processes are entirely reversible (unitary). So one might think that if the evolution of a system leads to concentration of amplitudes, then it should equally well lead to the reverse. But the crucial point is that while this may in principle be possible, it may essentially never happen in practice—just like classical reversible systems essentially never show behavior that goes against the Second Law of thermodynamics. As suggested by the main text, the details in the quantum measurement case are slightly more complicated—since to represent multiple outcomes measuring devices typically have to have the analogs of multiple equilibrium states. But the basic phenomena are ultimately very similar—and both are in effect based on the presence of microscopic randomness. (In a quantum system the randomness serves to give collections of complex numbers whose average is essentially always zero.)

This so-called decoherence approach was discussed in the 1930s, and finally began to become popular in the 1980s. But to make it work there needs to be some source of appropriate randomness. And almost without exception what has been assumed is that this must come through the first mechanism discussed in Chapter 7: that there is somehow randomness present in the environment that always gets into the system one is looking at. Various different specific mechanisms for this have been suggested, including ones based on ambient low-frequency photons, background quantum vacuum fluctuations and background spacetime metric fluctuations. (A somewhat related proposal involves quantum gravity effects in which irreversibility is assumed to be generated through analogs of the black hole processes mentioned in the previous note.) And indeed in recent practical experiments where simple pure quantum states have carefully been set up, they seem to be destroyed by randomness from the environment on timescales of at most perhaps microseconds. But this does not mean that in more complicated systems more characteristic of real measuring devices there may not be other sources of randomness that end up dominating.

One might imagine that a possibility would be the second mechanism for randomness from Chapter 7, based on ideas of chaos theory. For certainly in the standard formalism, quantum probability amplitudes are taken to be continuous quantities in which an arbitrary number of digits can be specified. But at least for a single particle, the Schrödinger equation is in all ways linear, and so it cannot support any kind of real sensitivity to initial conditions, or even to parameters. But when many particles are involved the situation can presumably be different, as it definitely can be in quantum field theory (see page 1061).

I suspect, however, that in fact the most important source of randomness in most cases will instead be the phenomenon of intrinsic randomness generation that I first discovered in systems like the rule 30 cellular automaton. Just like in so

many other areas, the emphasis on traditional mathematical methods has meant that for the most part fundamental studies have been made only on quantum systems that in the end turn out to have fairly simple behavior. Yet even within the standard formalism of quantum theory there are actually no doubt many closed systems that intrinsically manage to produce complex and seemingly random behavior even with very simple parameters and initial conditions. And in fact some clear signs of this were already present in studies of so-called quantum chaos in the 1980s—although most of the specific cases actually considered involved time-independent constraint satisfaction, not explicit time evolution. Curiously, what the Principle of Computational Equivalence suggests is that when quantum systems intrinsically produce apparent randomness they will in the end typically be capable of doing computations just as sophisticated as any other system—and in particular just as sophisticated as would be involved in conscious perception.

As a practical matter, mechanisms like intrinsic randomness generation presumably allow systems involving macroscopic numbers of particles to yield behavior in which interference becomes astronomically unlikely. But to reproduce the kind of exact reduction of probability amplitudes that is implied by the standard formalism of quantum theory inevitably requires taking the limit of an infinite system. Yet the Principle of Computational Equivalence suggests that the results of such a limit will typically be non-computable. (Using quantum field theory to represent infinite numbers of particles presumably cannot help; after appropriate analysis of the fairly sophisticated continuous mathematics involved, exactly the same computational issues should arise.)

It is often assumed that quantum systems should somehow easily be able to generate perfect randomness. But any sequence of bits one extracts must be deduced from a corresponding sequence of measurements. And certainly in practice—as mentioned on pages 303 and 970—correlations in the internal states of measuring devices between successive measurements will tend to lead to deviations from randomness. Whatever generates randomness and brings measuring devices back to equilibrium will eventually damp out such correlations. But insofar as measuring devices must formally involve infinite numbers of particles this process will formally require infinitely many steps. So this means that in effect an infinite computation is actually being done to generate each new bit. But with this amount of computation there are many ways to generate random bits. And in fact an infinite computation could even in principle produce algorithmic randomness (see page 1067) of the kind that is implicitly suggested by the traditional continuous mathematical formalism of quantum theory. So what this suggests is that there may in the end be no clear way to tell whether randomness is coming from an underlying quantum process that is being measured, or from the actual process of measurement. And indeed when it comes to more realistic finite measuring devices I would not be surprised if most of the supposed quantum randomness they measure is actually more properly attributed to intrinsic randomness generation associated with their internal mechanisms.

■ **Page 543 · Bell's inequalities.** In classical physics one can set up light waves that are linearly polarized with any given orientation. And if these hit polarizing ("anti-glare") filters whose orientation is off by an angle θ, then the waves transmitted will have intensity $Cos[\theta]^2$. In quantum theory the quantization of particle spin implies that any photon hitting a polarizing filter will always either just go through or be absorbed—so that in effect its spin measured relative to the orientation of the polarizer is either +1 or -1. A variety of atomic and other processes give pairs of photons that are forced to have total spin 0. And in what is essentially the Einstein-Podolsky-Rosen setup mentioned on page 1058 one can ask what happens if such photons are made to hit polarizers whose orientations differ by angle θ. In ordinary quantum theory, a straightforward calculation implies that the expected value of the product of the two measured spin values will be $-Cos[\theta]$. But now imagine instead that when each photon is produced it is assigned some "hidden variable" ϕ that in effect explicitly specifies the angle of its polarization. Then assume that a polarizer oriented at $0°$ will measure the spin of such a photon to have value $f[\phi]$ for some fixed function f. Now the expected value of the product of the two measured spin values is found just by averaging over ϕ as

```
Integrate[f[ϕ] f[θ - ϕ], {ϕ, 0, 2 π}]/(2 π)
```

A version of Bell's inequalities is then that this integral can decrease with θ no faster than $\theta/(2\pi) - 1$—as achieved when $f = Sign$. (In 3D ϕ must be extended to a sphere, but the same final result holds.) Yet as mentioned on page 1058, actual experiments show that in fact the decrease with θ is more rapid—and is instead consistent with the quantum theory result $-Cos[\theta]$. So what this means is that there is in a sense more correlation between measurements made on separated photons than can apparently be explained by the individual photons carrying any kind of explicit hidden property. (In the standard formalism of quantum theory this is normally explained by saying that the two photons can only meaningfully be considered as part of a single "entangled" state. Note that because of the probabilistic nature of the

correlations it turns out to be impossible to use them to do anything that would normally be considered communicating information faster than the speed of light.)

A basic assumption in deriving Bell's inequalities is that the choice of polarizer angle for measuring one photon is not affected by the choice of angle for the other. And indeed experiments have been done which try to enforce this by choosing the angles for the polarizers only just before the photons reach them—and too close in time for a light signal to get from one to the other. Such experiments again show violations of Bell's inequalities. But inevitably the actually devices that work out choices of polarizer angles must be in causal contact as part of setting up the experiment. And although it seems contrived, it is thus at least conceivable that with a realistic model for their time evolution such devices could end up operating in just such a way as to yield observed violations of Bell's inequalities.

Another way to get violations of Bell's inequalities is to allow explicit instantaneous propagation of information. But traditional models involving for example a background quantum potential again seem quite contrived, and difficult to generalize to relativistic cases. The approach I discuss in the main text is quite different, in effect using the idea that in a network model of space there can be direct connections between particles that do not in a sense ever have to go through ordinary intermediate points in space.

When set up for pairs of particles, Bell's inequalities tend just to provide numerical constraints on probabilities. But for triples of particles, it was noticed in the late 1980s that they can give constraints that force probabilities to be 0 or 1, implying that with the assumptions made, certain configurations of measurement results are simply impossible.

In quantum field theory the whole concept of measurement is much less developed than in quantum mechanics—not least because in field theory it is much more difficult to factor out subsystems, and so to avoid having to give explicit descriptions of measuring devices. But at least in axiomatic quantum field theory it is typically assumed that one can somehow measure expectation values of any suitably smeared product of field operators. (It is possible that these could be reconstructed from combinations of idealized scattering experiments). And to get a kind of analog of Bell's inequalities one can look at correlations defined by such expectation values for field operators at spacelike-separated points (too close in time for light signals to get from one to another). And it then turns out that even in the vacuum state the vacuum fluctuations that are present show nonzero such correlations—an analog of ordinary quantum mechanical entanglement. (In a non-interacting approximation these correlations turn out to be as large as is mathematically possible, but fall off exponentially outside the light cone, with exponents determined by the smallest particle mass or the measurement resolution.) In a sense, however, the presence of such correlations is just a reflection of the idealized way in which the vacuum state is set up—with each field mode determined all at once for the whole system.

Processes of Perception and Analysis

Defining the Notion of Randomness

▪ **Page 554 · Algorithmic information theory.** A description of a piece of data can always be thought of as some kind of program for reproducing the data. So if one could find the shortest program that works then this must correspond to the shortest possible description of the data—and in algorithmic information theory if this is no shorter than the data itself then the data is considered to be algorithmically random.

How long the shortest program is for a given piece of data will in general depend on what system is supposed to run the program. But in a sense the program will on the whole be as short as possible if the system is universal (see page 642). And between any two universal systems programs can differ in length by at most a constant: for one can always just add a fixed interpreter program to the programs for one system in order to make them run on the other system.

As mentioned in the main text, any data generated by a simple program can by definition never be algorithmically random. And so even though algorithmic randomness is often considered in theoretical discussions (see note below) it cannot be directly relevant to the kind of randomness we see in so many systems in this book—or, I believe, in nature.

If one considers all 2^n possible sequences (say of 0's and 1's) of length n then it is straightforward to see that most of them must be more or less algorithmically random. For in order to have enough programs to generate all 2^n sequences most of the programs one uses must themselves be close to length n. (In practice there are subtleties associated with the encoding of programs that make this hold only for sufficiently large n.) But even though one knows that almost all long sequences must be algorithmically random, it turns out to be undecidable in general whether any particular sequence is algorithmically random. For in general one can give no upper limit to how much computational effort one might have to expend in order to find out whether any given short program—after any number of steps—will generate the sequence one wants.

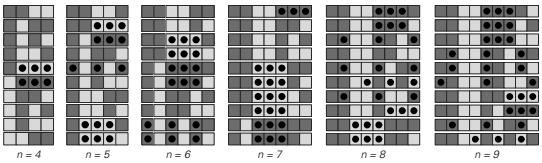
But even though one can never expect to construct them explicitly, one can still give formal descriptions of sequences that are algorithmically random. An example due to Gregory Chaitin is the digits of the fraction Ω of initial conditions for which a universal system halts (essentially a compressed version—with various subtleties about limits—of the sequence from page 1127 giving the outcome for each initial condition). As emphasized by Chaitin, it is possible to ask questions purely in arithmetic (say about sequences of values of a parameter that yield infinite numbers of solutions to an integer equation) whose answers would correspond to algorithmically random sequences. (See page 786.)

As a reduced analog of algorithmic information theory one can for example ask what the simplest cellular automaton rule is that will generate a given sequence if started from a single black cell. Page 1186 gives some results, and suggests that sequences which require more complicated cellular automaton rules do tend to look to us more complicated and more random.

▪ **History.** Randomness and unpredictability were discussed as general notions in antiquity in connection both with questions of free will (see page 1135) and games of chance. When probability theory emerged in the mid-1600s it implicitly assumed sequences random in the sense of having limiting frequencies following its predictions. By the 1800s there was extensive debate about this, but in the early 1900s with the advent of statistical mechanics and measure theory the use of ensembles (see page 1020) turned discussions of probability away from issues of randomness in individual sequences. With the development of statistical hypothesis testing in the early 1900s various tests for randomness were proposed (see page 1084). Sometimes these were claimed to have some kind of general significance, but mostly they were just viewed as simple practical methods. In many fields

outside of statistics, however, the idea persisted even to the 1990s that block frequencies (or flat frequency spectra) were somehow the only ultimate tests for randomness. In 1909 Emile Borel had formulated the notion of normal numbers (see page 912) whose infinite digit sequences contain all blocks with equal frequency. And in the 1920s Richard von Mises—attempting to capture the observed lack of systematically successful gambling schemes—suggested that randomness for individual infinite sequences could be defined in general by requiring that "collectives" consisting of elements appearing at positions specified by any procedure should show equal frequencies. To disallow procedures say specially set up to pick out all the infinite number of 1's in a sequence Alonzo Church in 1940 suggested that only procedures corresponding to finite computations be considered. (Compare page 1021 on coarse-graining in thermodynamics.) Starting in the late 1940s the development of information theory began to suggest connections between randomness and inability to compress data, but emphasis on *p Log[p]* measures of information content (see page 1071) reinforced the idea that block frequencies are the only real criterion for randomness. In the early 1960s, however, the notion of algorithmic randomness (see note above) was introduced by Gregory Chaitin, Andrei Kolmogorov and Ray Solomonoff. And unlike earlier proposals the consequences of this definition seemed to show remarkable consistency (in 1966 for example Per Martin-Löf proved that in effect it covered all possible statistical tests)—so that by the early 1990s it had become generally accepted as the appropriate ultimate definition of randomness. In the 1980s, however, work on cryptography had led to the study of some slightly weaker definitions of randomness based on inability to do cryptanalysis or make predictions with polynomial-time computations (see page 1089). But quite what the relationship of any of these definitions might be to natural science or everyday experience was never much discussed. Note that definitions of randomness given in dictionaries tend to emphasize lack of aim or purpose, in effect following the common legal approach of looking at underlying intentions (or say at physical construction of dice) rather than trying to tell if things are random from their observed behavior.

▪ **Inevitable regularities and Ramsey theory.** One might have thought that there could be no meaningful type of regularity that would be present in all possible data of a given kind. But through the development since the late 1920s of Ramsey theory it has become clear that this is not the case. As one example, consider looking for runs of m equally spaced squares of the same color embedded in sequences of black and white squares of length n. The pictures below show results with $m = 3$ for various n. For $n < 9$ there are always some sequences in which no runs of length 3 exist. But it turns out that for $n \geq 9$ every single possible sequence contains at least one run of length 3. For any m the same is true for sufficiently large n; it is known that $m = 4$ requires $n \geq 35$ and $m = 5$ requires $n \geq 178$. (In problems like this the analog of n often grows extremely rapidly with m.) If one has a sufficiently long sequence, therefore, just knowing that a run of equally spaced identical elements exists in it does not narrow down at all what the sequence actually is, and can so cannot ultimately be considered a useful regularity.



(Compare pattern-avoiding sequences on page 944.)

Defining Complexity

▪ **Page 557 · History.** There have been terms for complexity in everyday language since antiquity. But the idea of treating complexity as a coherent scientific concept potentially amenable to explicit definition is quite new: indeed this became popular only in the late 1980s—in part as a result of my own efforts. That what one would usually call complexity can be present in mathematical systems was for example already noted in the 1890s by Henri Poincaré in connection with the three-body problem (see page 972). And in the 1920s the issue of quantifying the complexity of simple mathematical formulas had come up in work on assessing statistical models (compare page 1083). By the 1940s general comments about biological, social and occasionally other systems being characterized by high complexity were common, particularly in connection with the cybernetics movement. Most often complexity seems to have been thought of as associated with the presence of large numbers of components with different types or behavior, and typically also with the presence of extensive interconnections or interdependencies. But occasionally—especially in some areas of social science—complexity was instead thought of as being characterized by somehow going beyond what human minds can handle. In the 1950s there was some discussion in pure mathematics of notions of complexity associated variously with sizes of axioms for logical theories, and with numbers of ways to satisfy such axioms. The development of information theory in the late 1940s—followed by the

discovery of the structure of DNA in 1953—led to the idea that perhaps complexity might be related to information content. And when the notion of algorithmic information content as the length of a shortest program (see page 1067) emerged in the 1960s it was suggested that this might be an appropriate definition for complexity. Several other definitions used in specific fields in the 1960s and 1970s were also based on sizes of descriptions: examples were optimal orders of models in systems theory, lengths of logic expressions for circuit and program design, and numbers of factors in Krohn-Rhodes decompositions of semigroups. Beginning in the 1970s computational complexity theory took a somewhat different direction, defining what it called complexity in terms of resources needed to perform computational tasks. Starting in the 1980s with the rise of complex systems research (see page 862) it was considered important by many physicists to find a definition that would provide some kind of numerical measure of complexity. It was noted that both very ordered and very disordered systems normally seem to be of low complexity, and much was made of the observation that systems on the border between these extremes—particularly class 4 cellular automata—seem to have higher complexity. In addition, the presence of some kind of hierarchy was often taken to indicate higher complexity, as was evidence of computational capabilities. It was also usually assumed that living systems should have the highest complexity—perhaps as a result of their long evolutionary history. And this made informal definitions of complexity often include all sorts of detailed features of life (see page 1178). One attempt at an abstract definition was what Charles Bennett called logical depth: the number of computational steps needed to reproduce something from its shortest description. Many simpler definitions of complexity were proposed in the 1980s. Quite a few were based just on changing $p_i\, Log[p_i]$ in the definition of entropy to a quantity vanishing for both ordered and disordered p_i. Many others were based on looking at correlations and mutual information measures—and using the fact that in a system with many interdependent and potentially hierarchical parts this should go on changing as one looks at more and more. Some were based purely on fractal dimensions or dimensions associated with strange attractors. Following my 1984 study of minimal sizes of finite automata capable of reproducing states in cellular automaton evolution (see page 276) a whole series of definitions were developed based on minimal sizes of descriptions in terms of deterministic and probabilistic finite automata (see page 1084). In general it is possible to imagine setting up all sorts of definitions for quantities that one chooses to call complexity. But what is most relevant for my purposes in this book is instead to find ways to capture everyday notions of complexity—and then to see how systems can produce these. (Note that since the 1980s there has been interest in finding measures of complexity that instead for example allow maintainability and robustness of software and management systems to be assessed. Sometimes these have been based on observations of humans trying to understand or verify systems, but more often they have just been based for example on simple properties of networks that define the flow of control or data—or in some cases on the length of documentation needed.) (The kind of complexity discussed here has nothing directly to do with complex numbers such as $\sqrt{-1}$ introduced into mathematics since the 1600s.)

Data Compression

▪ **Practicalities.** Data compression is important in making maximal use of limited information storage and transmission capabilities. One might think that as such capabilities increase, data compression would become less relevant. But so far this has not been the case, since the volume of data always seems to increase more rapidly than capabilities for storing and transmitting it. In the future, compression is always likely to remain relevant when there are physical constraints—such as transmission by electromagnetic radiation that is not spatially localized.

▪ **History.** Morse code, invented in 1838 for use in telegraphy, is an early example of data compression based on using shorter codewords for letters such as "e" and "t" that are more common in English. Modern work on data compression began in the late 1940s with the development of information theory. In 1949 Claude Shannon and Robert Fano devised a systematic way to assign codewords based on probabilities of blocks. An optimal method for doing this was then found by David Huffman in 1951. Early implementations were typically done in hardware, with specific choices of codewords being made as compromises between compression and error correction. In the mid-1970s, the idea emerged of dynamically updating codewords for Huffman encoding, based on the actual data encountered. And in the late 1970s, with online storage of text files becoming common, software compression programs began to be developed, almost all based on adaptive Huffman coding. In 1977 Abraham Lempel and Jacob Ziv suggested the basic idea of pointer-based encoding. In the mid-1980s, following work by Terry Welch, the so-called LZW algorithm rapidly became the method of choice for most general-purpose compression systems. It was used in programs such as PKZIP, as well as in hardware devices

such as modems. In the late 1980s, digital images became more common, and standards for compressing them emerged. In the early 1990s, lossy compression methods (to be discussed in the next section) also began to be widely used. Current image compression standards include: FAX CCITT 3 (run-length encoding, with codewords determined by Huffman coding from a definite distribution of run lengths); GIF (LZW); JPEG (lossy discrete cosine transform, then Huffman or arithmetic coding); BMP (run-length encoding, etc.); TIFF (FAX, JPEG, GIF, etc.). Typical compression ratios currently achieved for text are around 3:1, for line diagrams and text images around 3:1, and for photographic images around 2:1 lossless, and 20:1 lossy. (For sound compression see page 1080.)

■ **Page 560 · Number representations.** The sequence of 1's and 0's representing a number n are obtained as follows:

(a) *Unary.* `Table[0, {n}]`. (Not self-delimited.)

(b) *Ordinary base 2.* `IntegerDigits[n, 2]`. (Not self-delimited.)

(c) *Length prefixed.* Starting with an ordinary base 2 digit sequence, one prepends a unary specification of its length, then a specification of that length specification, and so on:

```
(Flatten[{Sign[-Range[1 - Length[#], 0]], #}] &)[
  Map[Rest, IntegerDigits[Rest[Reverse[NestWhileList[
        Floor[Log[2, #]] &, n + 1, # > 1 &]]], 2]]]
```

(d) *Binary-coded base 3.* One takes base 3 representation, then converts each digit to a pair of base 2 digits, handling the beginning and end of the sequence in a special way.

```
Flatten[IntegerDigits[
  Append[2 - With[{w = Floor[Log[3, 2 n]]},
     IntegerDigits[n - (3^(w+1) - 1)/2, 3, w]], 3], 2, 2]]
```

(e) *Fibonacci encoding.* Instead of decomposing a number into a sum of powers of an integer base, one decomposes it into a sum of Fibonacci numbers (see page 902). This decomposition becomes unique when one requires that no pair of 1's appear together.

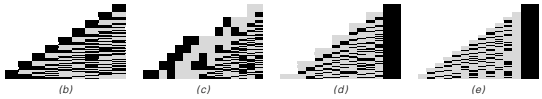
```
Apply[Take, RealDigits[(N[#, N[Log[10, #] + 3]] &)[
    n Sqrt[5] /GoldenRatio^2 + 1/2], GoldenRatio]]
```

The representations of all the first `Fibonacci[n] - 1` numbers can be obtained from (the version in the main text has `Rest[RotateLeft[Join[#, {0, 1}]]]` & applied)

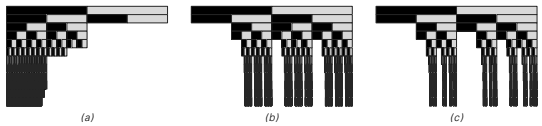
```
Apply[Join, Map[Last,
  NestList[{#[[2]], Join[Map[Join[{1, 0}, Rest[#]] &, #[[2]]],
      Map[Join[{1, 0}, #] &, #[[1]]]]} &, {{}, {{1}}}, n - 3]]]
```

■ **Lengths of representations.** (a) `n`, (b) `Floor[Log[2, n] + 1]`, (c) `Tr[FixedPointList[Max[0, Ceiling[Log[2, #]]] &, n + 2]] - n - 3`, (d) `2 Ceiling[Log[3, 2 n + 1]]`, (e) `Floor[Log[GoldenRatio, Sqrt[5] (n + 1/2)]]`. Large n approximations: (a) `n`, (b) `Log[2, n]`, (c) `Log[2, n] + Log[2, Log[2, n]] + …`, (d) `2 Log[3, n]`, (e) `Log[GoldenRatio, n]`.

Shown on a logarithmic scale, representations (b) through (e) (given here for numbers 1 through 500) all grow roughly linearly:



■ **Completeness.** If one successively reads 0's and 1's from an infinite sequence then the representations (c), (d) and (e) have the property that eventually one will always accumulate a valid representation for some number or another. The pictures below show which sequences of 0's and 1's correspond to complete numbers in these representations. Every vertical column is a possible sequence of 0's and 1's, and the column is shown to terminate when a complete number is obtained.



With an infinite random sequence of 0's and 1's, different number representations yield different distributions of sizes of numbers. Representation (b), for example, is more weighted towards large numbers, while (c) is more weighted towards small numbers. Maximal compression for a sequence of numbers with a particular distribution of sizes is obtained by choosing a representation that yields a matching such distribution. (See also page 949.)

■ **Practical computing.** Numbers used for arithmetic in practical computing are usually assumed to have a fixed length of, say, 32 bits, and thus do not need to be self-delimiting. In *Mathematica*, where integers can be of essentially any size, a representation closer to (b) above is used.

■ **Page 561 · Run-length encoding.** Data can be converted to run lengths by `Map[Length, Split[data]]`. Each number is then replaced by its representation.

With completely random input, the output will on average be longer by a factor `Sum[2^-(n+1) r[n], {n, 1, ∞}]` where `r[n]` is the length of the representation for n. For the Fibonacci encoding used in the main text, this factor is approximately 1.41028. (In base 2 this number has 1's essentially at positions `Fibonacci[n]`; as discussed on page 914, the number is transcendental.)

▪ **Page 563 · Huffman coding.** From a list *p* of probabilities for blocks, the list of codewords can be generated using

```
Map[Drop[Last[#], -1] &, Sort[
  Flatten[MapIndexed[Rule, FixedPoint[Replace[Sort[#],
      {{p0_, i0_}, {p1_, i1_}, pi___} → {{p0 + p1, {i0, i1}},
       pi}] &, MapIndexed[List, p]][[1, 2]], {-1}]]]] - 1
```

Given the list of codewords *c*, the sequence of blocks that occur in encoded data *d* can be uniquely reconstructed using

```
First[{{}, d} //. MapIndexed[
    {{r___}, Flatten[{#1, s___}]} → {{r, #2[[1]]}, {s}} &, c]]
```

Note that the encoded data can consist of any sequence of 0's and 1's. If all 2^b possible blocks of length *b* occur with equal probability, then the Huffman codewords will consist of blocks equivalent to the original ones. In an opposite extreme, blocks with probabilities $1/2$, $1/4$, $1/8$, … will yield codewords of lengths 1, 2, 3, …

In practical applications, Huffman coding is sometimes extended to allow the choice of codewords to be updated dynamically as more data is read.

▪ **Maximal block compression.** If one has data that consists of a long sequence of blocks, each of length *b*, and each independently chosen with probability *p[i]* to be of type *i*, then as argued by Claude Shannon in the late 1940s, it turns out that the minimum number of base 2 bits needed on average to represent each block in such a sequence is $h = -\text{Sum}[p[i] \text{Log}[2, p[i]], \{i, 2^b\}]$. If all blocks occur with an equal probability of 2^{-b}, then *h* takes on its maximum possible value of *b*. If only one block occurs with nonzero probability then *h* == 0. Following Shannon, the quantity *h* (whose form is analogous to entropy in physics, as discussed on page 1020) is often referred to as "information content". This name, however, is very misleading. For certainly *h* does not in general give the length of the shortest possible description of the data; all it does is to give the shortest length of description that is obtained by treating successive blocks as if they occur with independent probabilities. With this assumption one then finds that maximal compression occurs if a block of probability *p[i]* is represented by a codeword of length $-\text{Log}[2, p[i]]$. Huffman coding with a large number of codewords will approach this if all the *p[i]* are powers of 1/2. (The self-delimiting of codewords leads to deviations for small numbers of codewords.) For *p[i]* that are not powers of 1/2, non-integer length codewords would be required. The method of arithmetic coding provides an alternative in which the output does not consist of separate codewords concatenated together. (Compare algorithmic information content discussed on pages 554 and 1067.)

▪ **Arithmetic coding.** Consider dividing the interval from 0 to 1 into a succession of bins, with each bin having a width equal to the probability for some sequence of blocks to occur. The idea of arithmetic coding is to represent each such bin by the digit sequence of the shortest number within the bin—after trailing zeros have been dropped. For any sequence *s* this can be done using

```
Module[{c, m = 0},
 Map[c[#] = {m, m += Count[s, #]/Length[s]} &, Union[s]];
 Function[x, (First[RealDigits[2^# Ceiling[2^-# Min[x]],
    2, -#, -1]] &)[Floor[Log[2, Max[x] - Min[x]]]][
  Fold[(Max[#1] - Min[#1]) c[#2] + Min[#1] &, {0, 1}, s]]]
```

Huffman coding of a sequence containing a single 0 block together with *n* 1 blocks will yield output of length about *n*; arithmetic coding will yield length about *Log[n]*. Compression in arithmetic coding still relies, however, on unequal block probabilities, just like in Huffman coding. Originally suggested in the early 1960s, arithmetic coding reemerged in the late 1980s when high-speed floating-point computation became common, and is occasionally used in practice.

▪ **Page 565 · Pointer-based encoding.** One can encode a list of data *d* by generating pointers to the longest and most recent copies of each subsequence of length at least *b* using

```
PEncode[d_, b_ : 4] := Module[{i, a, u, v},
  i = 2; a = {First[d]}; While[i ≤ Length[d], {u, v} =
     Last[Sort[Table[{MatchLength[d, i, j], j}, {j, i - 1}]]];
   If[u ≥ b, AppendTo[a, p[i - v, u]]; i += u,
    AppendTo[a, d[[i]]]; i++]]; a]
MatchLength[d_, i_, j_] := With[{m = Length[d] - i}, Catch[
  Do[If[d[[i + k]] =!= d[[j + k]], Throw[k]], {k, 0, m}]; m + 1]]
```

The process of encoding can be made considerably faster by keeping a dictionary of previously encountered subsequences. One can reproduce the original data using

```
PDecode[a_] := Module[{d = Flatten[
   a /. p[j_, r_] :→ Table[p[j], {r}]]}, Flatten[MapIndexed[
   If[Head[#1] === p, d[[#2]] = d[[#2 - First[#1]]], #1] &, d]]]
```

To get a representation purely in terms of 0 and 1, one can use a self-delimiting representation for each integer that appears. (Knowing the explicit representation one could then determine whether each block would be shorter if encoded literally or using a pointer.) The encoded version of a purely repetitive sequence of length *n* has a length that grows like *Log[n]*. The encoded version of a purely nested sequence grows like $\text{Log}[n]^2$. The encoded version of a sufficiently random sequence grows like *n* (with the specific encoding used in the text, the length is about $2n$). Note that any sequence of 0's and 1's corresponds to the beginning of the encoding for some sequence or another.

It is possible to construct sequences whose encoded versions grow roughly like fractional powers of *n*. An example is the sequence *Table[Append[Table[0, {r}], 1], {r, s}]* whose encoded version grows like $\sqrt{n}\ \text{Log}[n]$. Cyclic tag systems often seem to produce sequences whose encoded versions grow like fractional

powers of n. Sequences produced by concatenation sequences are not typically compressed by pointer encoding.

With completely random input, the probability that the length b subsequence which begins at element n is a repeat of a previous subsequence is roughly $1-(1-2^{-b})^{n-1}$. The overall fraction of a length n input that consists of repeats of length at least b is greater than $1-2^b/n$ and is essentially

```
1 - Sum[(1 - 2^-b)^i Product[1 + (1 - 2^-b)^j - (1 - 2^(-b-1))^j,
    {j, i - b + 1, i - 1}], {i, b, n - b}]/(n - 2 b + 1)
```

■ **LZW algorithms.** Practical implementations of pointer-based encoding can maintain only a limited dictionary of possible repeats. Various schemes exist for optimizing the construction, storage and rewriting of such dictionaries.

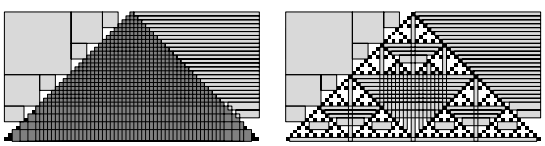
■ **Page 568 · Recursive subdivision.** In one dimension, encoding can be done using

```
Subdivide[a_] := Flatten[
    If[Length[a] == 2, a, If[Apply[SameQ, a], {1, First[a]},
        {0, Map[Subdivide, Partition[a, Length[a]/2]]}]]]
```

In n dimensions, it can be done using

```
Subdivide[a_, n_] := With[{s = Table[1, {n}]}, Flatten[
    If[Dimensions[a] == 2 s, a, If[Apply[SameQ, Flatten[a]],
        {1, First[Flatten[a]]}, {0, Map[Subdivide[#, n] &,
            Partition[a, 1/2 Length[a] s], {n}]}]]]]
```

■ **2D run-length encoding.** A simple way to generalize run-length encoding to two dimensions is to scan data one row after another, always finding the largest rectangle of uniform color that starts at each particular point. The pictures below show regions with an area of more than 10 cells found in this way. The presence of so many thin and overlapping regions prevents good compression.



2D run-length encoding can also be done by scanning the data according to a more complicated space-filling curve, of the kind discussed on page 893.

Irreversible Data Compression

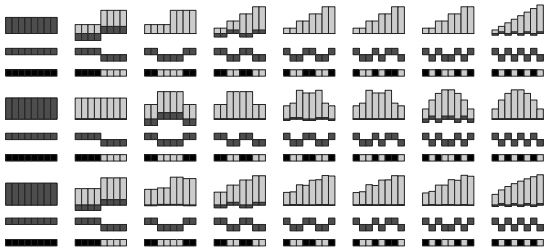
■ **History.** The idea of creating sounds by adding together pure tones goes back to antiquity. At a mathematical level, following work by Joseph Fourier around 1810 it became clear by the mid-1800s how any sufficiently smooth function could be decomposed into sums of sine waves with frequencies corresponding to successive integers. Early telephony and sound recording in the late 1800s already used the idea of compressing sounds by dropping high- and low-frequency components. From the early days of television in the 1950s, some attempts were made to do similar kinds of compression for images. Serious efforts in this direction were not made, however, until digital storage and processing of images became common in the late 1980s.

■ **Orthogonal bases.** The defining feature of a set of basic forms is that it is complete, in the sense that any piece of data can be built up by adding the basic forms with appropriate weights. Most sets of basic forms used in practice also have the feature of being orthogonal, which turns out to make it particularly easy to work out the weights for a given piece of data. In 1D, a basic form `a[[i]]` is just a list. Orthogonality is then the property that `a[[i]] . a[[j]] == 0` for all $i \neq j$. And when this property holds, the weights are given essentially just by `data . a`.

The concept of orthogonal bases was historically worked out first in the considerably more difficult case of continuous functions. Here a typical orthogonality property is `Integrate[f[r, x] f[s, x], {x, 0, 1}] == KroneckerDelta[r, s]`. As discovered by Joseph Fourier around 1810, this is satisfied for basis functions such as $\text{Sin}[2\,n\,\pi\,x]/\sqrt{2}$.

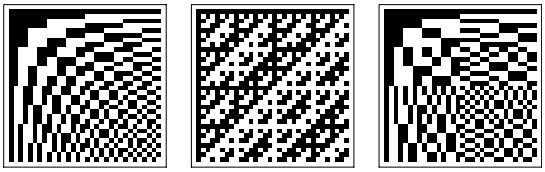
■ **Page 573 · Walsh transforms.** The basic forms shown in the main text are 2D Walsh functions—represented as ± 1 matrices. Each collection of such functions can be obtained from lists of vectors representing 1D Walsh functions by using `Outer[Outer[Times, ##] &, b, b, 1, 1]`, or equivalently `Map[Transpose, Map[# b &, b, {2}]]`.

The pictures below show how 1D arrays of data values can be built up by adding together 1D Walsh functions. At each step the Walsh function used is given underneath the array of values obtained so far.



The components of the vectors for 1D Walsh functions can be ordered in many ways. The pictures below show the

complete matrices of basis vectors obtained with three common orderings.



The matrices for size $n = 2^s$ can be obtained from

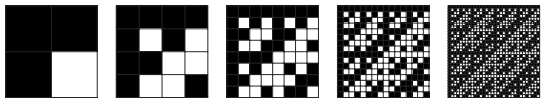
```
Nest[Apply[Join, f[{Map[Flatten[Map[{#, #} &, #]] &, #],
    Map[Flatten[Map[{#, -#} &, #]] &, g[#]]]]] &, {{1}}, s]
```

with (a) `f = Identity`, `g = Reverse`, (b) `f = Transpose`, `g = Identity`, and (c) `f = g = Identity`. (a) is used in the main text. Known as sequency order, it has the property that each row involves one more change of color than the previous row. (b) is known as natural or Hadamard order. It exhibits a nested structure, and can be obtained as in the pictures below from the evolution of a 2D substitution system, or equivalently from a Kronecker product as in

```
Nest[Flatten2D[Map[# {{1, 1}, {1, -1}} &, #, {2}]] &, {{1}}, s]
```

with

```
Flatten2D[a_] :=
  Apply[Join, Apply[Join, Map[Transpose, a], {2}]]
```



(c) is known as dyadic or Paley order. It is related to (a) by Gray code reordering of the rows, and to (b) by reordering according to (see page 905)

```
BitReverseOrder[a_] :=
  With[{n = Length[a]}, a[[Map[FromDigits[Reverse[#], 2] &,
      IntegerDigits[Range[0, n - 1], 2, Log[2, n]]] + 1]]
```

It is also given by

```
Array[Apply[Times, (-1)^(IntegerDigits[#1, 2, s]
      Reverse[IntegerDigits[#2, 2, s]])] &, 2^{s, s}, 0]
```

where (b) is obtained simply by dropping the `Reverse`.

Walsh functions can correspond to nested sequences. The function at position $2/3\,(1 + 4^{(-(\mathrm{Floor}[s/2] + 1/2))})\,2^s$ in basis (a), for example, is exactly the Thue-Morse sequence (with 0 replaced by -1) from page 83.

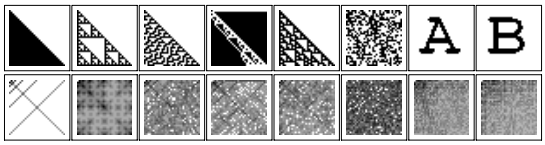
Given the matrix m of basis vectors, the Walsh transform is simply `data . m`. Direct evaluation of this for length n takes n^2 steps. However, the nested structure of m in natural order allows evaluation in only about $n \log[n]$ steps using

```
Nest[Flatten[Transpose[Partition[#, 2] . {{1, 1}, {1, -1}}]] &,
  data, Log[2, Length[data]]]
```

This procedure is similar to the fast Fourier transform discussed below. Transforms of 2D data are equivalent to 1D transforms of flattened data.

Walsh functions were used by electrical engineers such as Frank Fowle in the 1890s to find transpositions of wires that minimized crosstalk; they were introduced into mathematics by Joseph Walsh in 1923. Raymond Paley introduced the dyadic basis in 1932. Mathematical connections with harmonic analysis of discrete groups were investigated from the late 1940s. In the 1960s, Walsh transforms became fairly widespread in discrete signal and image processing.

▪ **Page 575 · Walsh spectra.** The arrays of absolute values of weights of basic forms for successive images are as follows:

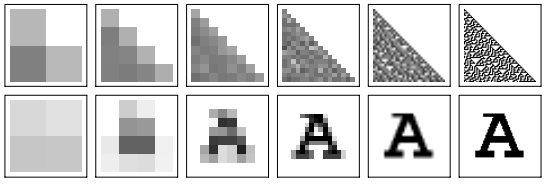


▪ **Hadamard matrices.** Hadamard matrices are $n \times n$ matrices with elements -1 and +1, whose rows are orthogonal, so that `m . Transpose[m] == n IdentityMatrix[n]`. The matrices used in Walsh transforms are special cases with $n = 2^s$. There are thought to be Hadamard matrices with every size $n = 4k$ (and for $n > 2$ no other sizes are possible); the number of distinct such matrices for each k up to 7 is 1, 1, 1, 5, 3, 60, 487. The so-called Paley family of Hadamard matrices for $n = 4k = p + 1$ with p prime are given by

```
PadLeft[Array[JacobiSymbol[#2 - #1, n - 1] &, {n, n} - 1] -
    IdentityMatrix[n - 1], {n, n}, 1]
```

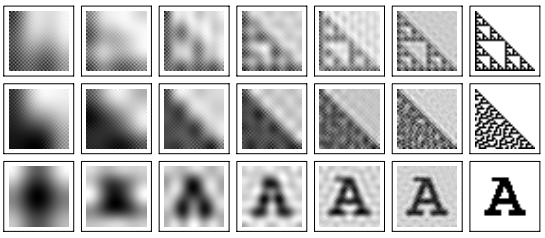
Originally introduced by Jacques Hadamard in 1893 as the matrices with elements $\mathrm{Abs}[a] \le 1$ which attain the maximal possible determinant $\pm n^{n/2}$, Hadamard matrices appear in various combinatorial problems, particularly design of exhaustive combinations of experiments and Reed-Muller error-correcting codes.

▪ **Image averaging.** Walsh functions yield significantly better compression than simple successive averaging of 2×2 blocks of cells, as shown below.



■ **Practical image compression.** Two basic phenomena contribute to our ability to compress images in practice. First, that typical images of relevance tend to be far from random—indeed they often involve quite limited numbers of distinct objects. And second, that many fine details of images go unnoticed by the human visual system (see the next section).

■ **Fourier transforms.** In a typical Fourier transform, one uses basic forms such as *Exp[i π r x/n]* with *r* running from 1 to *n*. The weights associated with these forms can be found using *Fourier*, and given these weights the original data can also be reconstructed using *InverseFourier*. The pictures below show what happens in such a so-called discrete cosine transform when different fractions of the weights are kept, and others are effectively set to zero. High-frequency wiggles associated with the so-called Gibbs phenomenon are typical near edges.



Fourier[data] can be thought of as multiplication by the $n \times n$ matrix *Array[Exp[2 π i #1 #2/n] &, {n, n}, 0]*. Applying *BitReverseOrder* to this matrix yields a matrix which has an essentially nested form, and for size $n = 2^s$ can be obtained from

```
Nest[With[{c = BitReverseOrder[Range[0, Length[#] - 1]/
      Length[#]]}, Flatten2D[MapIndexed[#1 {{1, 1},
       {1, -1} (-1)^c[[Last[#2]]]} &, #, {2}]]] &, {{1}}, s]
```

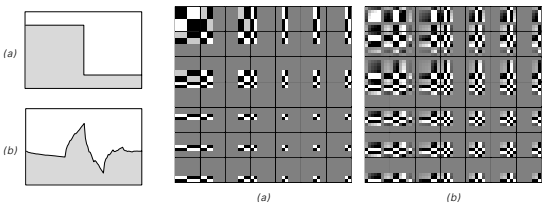
Using this structure, one obtains the so-called fast Fourier transform which operates in *n Log[n]* steps and is given by

```
With[{n = Length[data]}, Fold[Flatten[Map[With[
    {k = Length[#]/2}, {{1, 1}, {1, -1}} . {Take[#, k], Drop[
     #, k] (-1)^(Range[0, k - 1]/k)}] &, Partition[##]]] &,
  BitReverseOrder[data], 2^Range[Log[2, n]]]/√n ]
```

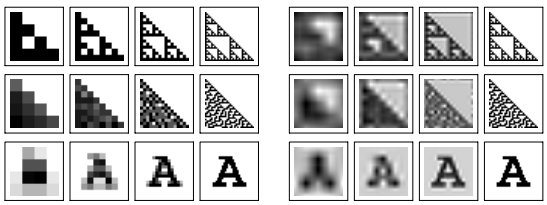
(See also page 1080.)

■ **JPEG compression.** In common use since the early 1990s JPEG compression works by first assigning color values to definite bins, then applying a discrete Fourier cosine transform, then applying Huffman encoding to the resulting weights. The "quality" of the image is determined by how many weights are kept; a typical default quality factor, used say by *Export* in *Mathematica*, is 75.

■ **Wavelets.** Each basic form in an ordinary Walsh or Fourier transform has nonzero elements spread throughout. With wavelets the elements are more localized. As noted in the late 1980s basic forms can be set up by scaling and translating just a single appropriately chosen underlying shape. The (a) Haar and (b) Daubechies wavelets $\psi[x]$ shown below both have the property that the basic forms $2^{m/2}\, \psi[2^m x - n]$ (whose 2D analogs are shown as on page 573) are orthogonal for every different m and n.



The pictures below show images built up by keeping successively more of these basic forms. Sharp edges have fewer wiggles than with Fourier transforms.



■ **Sound compression.** See page 1080.

Visual Perception

■ **Color vision.** The three types of color-sensitive cone cells on the human retina each have definite response curves as a function of wavelength. The perceived color of light with a given wavelength distribution is basically determined by the three numbers obtained by integrating these responses. For any wavelength distribution it turns out that if one scales these numbers to add up to one, then the chromaticity values obtained must lie within a certain region. Mixing n specific colors in different proportions allows one to reach any point in an n-cornered polytope. For $n = 3$ this polytope comes close to filling the region of all possible colors, but for no n can it completely fill it—which is why practical displays and printing processes can produce only limited ranges of colors.

An important observation, related to the fact that limitations in color ranges are usually not too troublesome, is that the perceived colors of objects stay more or less constant even when viewed in very different lighting, corresponding to very different wavelength distributions. In recent years it has become clear that the origin of this phenomenon is that

beyond the original cone cells, most color-sensitive cells in our visual system respond not to absolute color levels, but instead to differences in color levels at slightly different positions. (Responses to nearby relative values rather than absolute values seem to be common in many forms of human perception.)

The fact that white light is a mixture of colors was noticed by Isaac Newton in 1704, and it became clear in the course of the 1700s that three primaries could reproduce most colors. Thomas Young suggested in 1802 that there might be three types of color receptors in the eye, but it was not until 1959 that these were actually identified—though on the basis of perceptual experiments, parametrizations of color space were already well established by the 1930s. While humans and primates normally have three types of cone cells, it has been found that other mammals normally have two, while birds, reptiles and fishes typically have between 3 and 5.

▪ **Nerve cells.** In the retina and the brain, nerve cells typically have an irregular tree-like structure, with between a few and a few thousand dendrites carrying input signals, and one or more axons carrying output signals. Nerve cells can respond on timescales of order milliseconds to changes in their inputs by changing their rate of generating output electrical spikes. As has been believed since the 1940s, most often nerve cells seem to operate at least roughly by effectively adding up their inputs with various positive or negative weights, then going into an excited state if the result exceeds some threshold. The weights seem to be determined by detailed properties of the synapses between nerve cells. Their values can presumably change to reflect certain aspects of the activity of the cell, thus forming a basis for memory (see page 1102). In organisms with a total of only a few thousand nerve cells, each individual cell typically has definite connections and a definite function. But in humans with perhaps 100 billion nerve cells, the physical connections seem quite haphazard, and most nerve cells probably develop their function as a result of building up weights associated with their actual pattern of behavior, either spontaneous or in response to external stimuli.

▪ **The visual system.** Connected to the 100 million or so light-sensitive photoreceptor cells on the retina are roughly two layers of nerve cells, with various kinds of cross-connections, out of which come the million fibers that form the optic nerve. After essentially one stop, most of these go to the primary visual cortex at the back of the brain, which itself contains more than 100 million nerve cells. Physical connections between nerve cells have usually been difficult to map. But starting in the 1950s it became possible to record electrical activity in single cells, and from this the discovery was made that many cells respond to rather specific visual stimuli. In the retina, most common are center-surround cells, which respond when there is a higher level of light in the center of a roughly circular region and a lower level outside, or vice versa. In the first few layers of the visual cortex about half the cells respond to elongated versions of similar stimuli, while others seem sensitive to various forms of change or motion. In the fovea at the center of the retina, a single center-surround cell seems to get input from just a few nearby photoreceptors. In successive layers of the visual cortex cells seem to get input from progressively larger regions. There is a very direct mapping of positions on the retina to regions in the visual cortex. But within each region there are different cells responding to stimuli at different angles, as well as to stimuli from different eyes. Cells with particular kinds of responses are usually found to be arranged in labyrinthine patterns very much like those shown on page 427. And no doubt the processes which produce these patterns during the development of the organism can be idealized by simple 2D cellular automata. Quite what determines the pattern of illumination to which a given cell will respond is not yet clear, although there is some evidence that it is the result of adaptation associated with various kinds of test inputs. Since the late 1970s, it has been common to assume that the response of a cell can be modelled by derivatives of Gaussians such as those shown below, or perhaps by Gabor functions given by products of trigonometric functions and Gaussians. Experiments have determined responses to these and other specific stimuli, but inevitably no experiment can find all the stimuli to which a cell is sensitive.

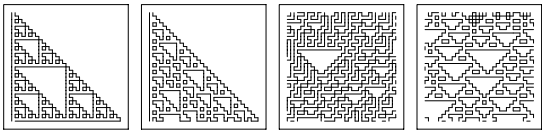
The visual systems of a number of specific higher and lower organisms have now been studied, and despite a few differences (such as cross-connections being behind the photoreceptors on the retinas of octopuses and squids, but in front in most higher animals), the same general features are usually seen. In lower organisms, there tend to be fewer layers of cells, with individual cells more specialized to particular visual stimuli of relevance to the organism.

▪ **Feedback.** Most of the lowest levels of visual processing seem to involve only signals going successively from one layer in the eye or brain to the next. But presumably there is at least some feedback to previous layers, yielding in effect iteration of rules like the ones used in the main text. The

resulting evolution process is likely to have attractors, potentially explaining the fact that in images such as "Magic Eye" random dot stereograms features can pop out after several seconds or minutes of scrutiny, even without any conscious effort.

▪ **Scale invariance.** In a first approximation our recognition of objects does not seem to be much affected by overall size or overall light level. For light level—as with color constancy—this is presumably achieved by responding only to differences between levels at different positions. Probably the same effect contributes to scale invariance by emphasizing only edges and corners. And if one is looking at objects like letters, it helps that one has learned them at many different sizes. But also similar cells most likely receive inputs from regions with a range of different sizes on the retina—making even unfamiliar textures seem the same over at least a certain range of scales. When viewed at a normal reading distance of 12 inches each square in the picture on page 578 covers a region about 5 cells across on the retina. With good lighting and good eyesight the textures in the picture can still be distinguished at a distance of 5 feet, where each square covers only one cell. But if the picture is enlarged by a factor of 3 or more then at normal reading distance it can become difficult to distinguish the textures—perhaps because the squares cover regions larger than the templates used at the lowest levels in our visual system.

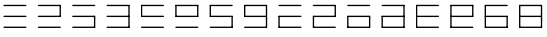
▪ **History.** Ever since antiquity the visual arts have yielded practical schemes and sometimes also fairly abstract frameworks for determining what features of images will have what impact. In fact, even in prehistoric times it seems to have been known, for example, that edges are often sufficient to communicate visual forms, as in the pictures below.



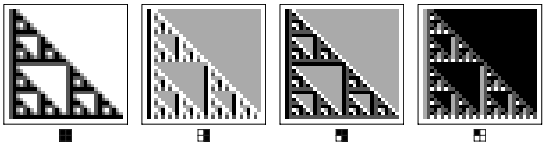
Visual perception has been used for centuries as an example in philosophical discussions about the nature of experience. Traditional mathematical methods began to be applied to it in the second half of the 1800s, particularly through the development of psychophysics. Studies of visual illusions around the end of the 1800s raised many questions that were not readily amenable to numerical measurement or traditional mathematical analysis, and this led in part to the Gestalt approach to psychology which attempted to formulate various global principles of visual perception.

In the 1940s and 1950s, the idea emerged that visual images might be processed using arrays of simple elements. At a largely theoretical level, this led to the perceptron model of the visual system as a network of idealized neurons. And at a practical level it also led to many systems for image processing (see below), based essentially on simple cellular automata (see page 928). Such systems were widely used by the end of the 1960s, especially in aerial reconnaissance and biomedical applications.

Attempts to characterize human abilities to perceive texture appear to have started in earnest with the work of Bela Julesz around 1962. At first it was thought that the visual system might be sensitive only to the overall autocorrelation of an image, given by the probability that randomly selected points have the same color. But within a few years it became clear that images could be constructed—notably with systems equivalent to additive cellular automata (see below)—that had the same autocorrelations but looked completely different. Julesz then suggested that discrimination between textures might be based on the presence of "textons", loosely defined as localized regions like those shown below with some set of distinct geometrical or topological properties.



In the 1970s, two approaches to vision developed. One was largely an outgrowth of work in artificial intelligence, and concentrated mostly on trying to use traditional mathematics to characterize fairly high-level perception of objects and their geometrical properties. The other, emphasized particularly by David Marr, concentrated on lower-level processes, mostly based on simple models of the responses of single nerve cells, and very often effectively applying *ListConvolve* with simple kernels, as in the pictures below.



In the 1980s, approaches based on neural networks capable of learning became popular, and attempts were made in the context of computational neuroscience to create models combining higher- and lower-level aspects of visual perception.

The basic idea that early stages of visual perception involve extraction of local features has been fairly clear since the 1950s, and researchers from a variety of fields have invented and reinvented implementations of this idea many times. But mainly through a desire to use traditional mathematics, these

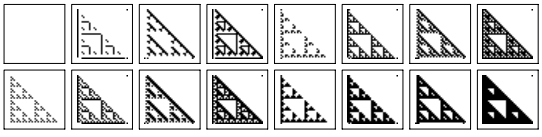
implementations have tended to be implicitly restricted to using elements with various linearity properties—typically leading to rather unconvincing results. My model is closer to what is often done in practical image processing, and apparently to how actual nerve cells work, and in effect assumes highly nonlinear elements.

▪ **Page 581 · Implementation.** The exact matches for a template σ in data containing elements 0 and 1 can be obtained from

```
Sign[ListCorrelate[2 σ - 1, data] - Count[σ, 1, 2]] + 1
```

▪ **Testing the model.** Although it is difficult to get good systematic data, the many examples I have tried indicate that the levels of discrimination between textures that we achieve with our visual system agree remarkably well with those suggested by my simple model. A practical issue that arises is that if one repeatedly tries experiments with the same set of textures, then after a while one learns to discriminate these particular textures better. Shifting successive rows or even just making an overall rotation seems, however, to avoid this effect.

▪ **Related models.** Rather than requiring particular templates to be matched, one can consider applying arbitrary cellular automaton rules. The pictures below show results from a single step of the 16 even-numbered totalistic 5-neighbor rules. The results are surprisingly easy to interpret in terms of feature extraction.



▪ **Image processing.** The release of programs like Photoshop in the late 1980s made image processing operations such as smoothing, sharpening and edge detection widely available on general-purpose computers. Most of these operations are just done by applying `ListConvolve` with simple kernels. (Even before computers, such convolutions could be done using the fact that diffraction of light effectively performs Fourier transforms.) Ever since the 1960s all sorts of schemes for nonlinear processing of images have been discussed and used in particular communities. An example originally popular in the earth and environmental sciences is so-called mathematical morphology, based on "dilation" of data consisting of 0's and 1's with a "structuring element" σ according to `Sign[ListConvolve[σ, data, 1, 0]]` (as well as the dual operation of "erosion"). Most schemes like this can ultimately be thought of as picking out templates or applying simple cellular automaton rules.

▪ **Real textures.** The textures I consider in the main text are all based on arrays of discrete black and white squares. One can also consider textures associated, say, with surface roughness of physical objects. Models of these are often needed for realistic computer graphics. Common approaches are to assume that the surfaces are random with some frequency spectrum, or can be generated as fractals using substitution systems with random parameters. In recent times, models based on wavelets have also been used.

▪ **Statistical methods.** Even though they do not appear to correspond to how the human visual system works, statistical methods are often used in trying to discriminate textures automatically. Correlations, conditional entropies and fractal dimensions are commonly computed. Often it is assumed that different parts of a texture are statistically independent, so that the texture can be characterized by probabilities for local patterns, as in a so-called Markov random field or generalized autoregressive moving average (ARMA) process.

▪ **Camouflage.** On both animals and military vehicles it is often important to have patterns that cannot be distinguished from a background by the visual systems of predators. And in most cases this is presumably best achieved by avoiding differences in densities of certain local features. Note that in a related situation almost any fairly random overlaid pattern containing many local features can successfully be used to mask the contents of a paper envelope.

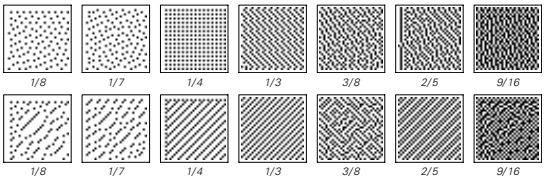
▪ **Halftoning.** In printed books like this one, gray levels are usually obtained by printing small dots of black with varying sizes. On displays consisting of fixed arrays of pixels, gray levels must be obtained by having only a certain density of pixels be black. One way to achieve this is to break the array into $2^n \times 2^n$ blocks, then successively to fill in pixels in each block until the appropriate gray level is reached, as in the pictures below, in an order given for example by

```
Nest[
  Flatten2D[{{4# + 0, 4# + 2}, {4# + 3, 4# + 1}}] &, {{0}}, n]
```

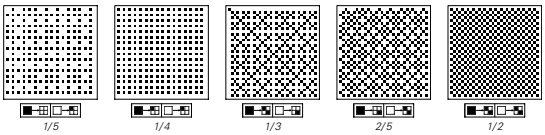
An alternative to this so-called ordered dither approach is the Floyd-Steinberg or error-diffusion method invented in 1976. This scans sequentially, accumulating and spreading total gray level in the data, then generating a black pixel whenever a threshold is exceeded. The method can be implemented using

```
Module[{a = Flatten[data], r, s},
 {r, s} = Dimensions[data]; Partition[Do[
   a[[i + {1, s - 1, s, s + 1}]] += m (a[[i]] - If[a[[i]] < 1/2, 0, 1]),
   {i, r s - s - 1}]; Map[If[# < 1/2, 0, 1] &, a], s]]
```

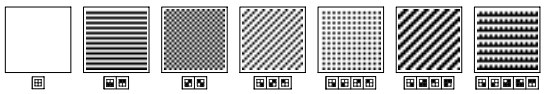
In its original version *m = {7, 3, 5, 1}/16*, as in the first row of pictures below. But even with *m = {1, 0, 1, 0}/2* the method generates fairly random patterns, as in the second row below. (Note that significantly different results can be obtained if different boundary conditions are used for each row.)



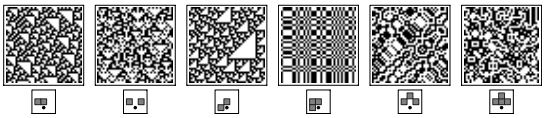
To give the best impression of uniform gray, one must in general minimize features detected by the human visual system. One simple way to do this appears to be to use nested patterns like the ones below.



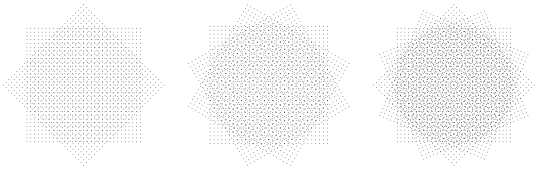
▪ **Generating textures.** As discussed on page 217, it is in general difficult to find 2D patterns which at all points match some definite set of templates. With 2×2 templates, there turn out to be just 7 minimal such patterns, shown below. Constructing patterns in which templates occur with definite densities is also difficult, although randomized iterative schemes allow some approximation to be obtained.



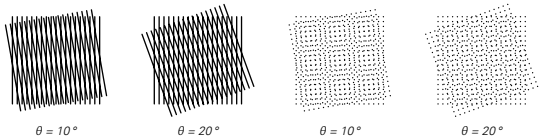
One-dimensional cellular automata are especially convenient generators of distinctive textures. Indeed, as was noticed around 1980, generalizations of additive rules involving cells in different relative locations can produce textures with similar statistics, but different visual appearance, as shown below. (All the examples shown turn out to correspond to ordinary, sequential and reversible cellular automata seen elsewhere in this book.) (See also page 1018.)



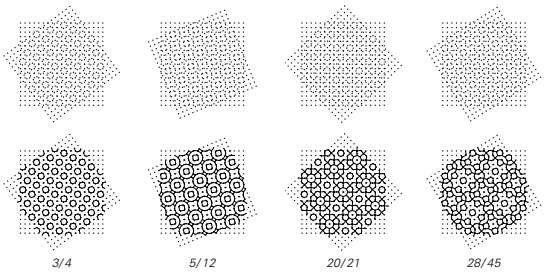
▪ **Moire patterns.** The pictures below show moire patterns formed by superimposing grids of points at different angles. Our visual system does not immediately perceive the grids, but instead mainly picks up features formed from local arrangements of dots. The second picture below is similar to patterns of halftone screens visible in 4-color printing under a magnifying glass.



In the first two pictures below, bands with spacing *1/2 Csc[θ/2]* are visible wherever lines cross. In the second two pictures there is also an apparent repetitive pattern with approximately the same repetition period.

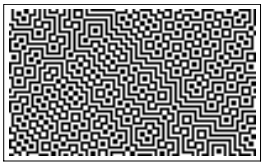


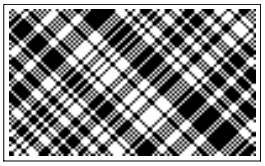
The patterns are exactly repetitive only when *Tan[θ] == u/v*, where *u* and *v* are elements of a primitive Pythagorean triple (so that *u*, *v* and *Sqrt[u² + v²]* are all integers, and *GCD[u, v] == 1*). This occurs when $u = r^2 - s^2$, $v = 2 r s$ (see page 945), and in this case the minimum displacement that leaves the whole pattern unchanged is *{s, r}*.



The second row of pictures illustrates what happens if points closer than distance $1/\sqrt{2}$ are joined. The results appear to capture at least some of the features picked out by our visual system.

▪ **Perception and presentation.** In writing this book it has been a great challenge to find graphical representations that make the behavior of systems as clear as possible for the purposes of human visual perception. Even small changes in representation can greatly affect what properties are noticed. As a simple example, the pictures below are identical, except for the fact that the colors of cells on alternate rows have been reversed.





Auditory Perception

▪ **Sounds.** The human auditory system is sensitive to sound at frequencies between about 20 Hz and 20 kHz. Middle A on a piano typically corresponds to a frequency of 440 Hz. Each octave represents a change in frequency by a factor of two. In western music there are normally 12 notes identified within an octave. These differ in frequency by successive factors of roughly $2^{1/12}$—with different temperament schemes using different rational approximations to powers of this quantity.

The perceived character of a sound seems to depend most on the frequencies it contains, but also to be somewhat affected by the way its intensity ramps up with time, as well as the way frequencies change during this ramp up. Many musical instruments produce sound by vibrating strings or air in cylindrical or conical tubes, and in these cases, there is one main frequency, together with roughly equally spaced overtones. In percussion instruments, the spectrum of frequencies is usually much more complicated. In speech, vowels and voiced consonants tend to be characterized by the lowest two or three frequencies of the mouth. In nature, processes such as fluid turbulence and fracture yield a broad spectrum of frequencies. In speech, letters like "s" also yield broad spectra, presumably because they involve fluid turbulence.

Any sound can be specified by giving its amplitude or waveform as a function of time. $Sin[\omega t]$ corresponds to a pure tone. Other simple mathematical functions can also yield distinctive sounds. FM synthesis functions such as $Sin[\omega (t + a\, Sin[b\, t])]$ can be made to sound somewhat like various musical instruments, and indeed were widely used in early synthesizers.

▪ **Auditory system.** Sound is detected by the motion it causes in hair cells in the cochlea of the inner ear. When vibrations of a particular frequency enter the cochlea an active process involving hair cells causes the vibrations to be concentrated at a certain distance down the cochlea. To a good approximation this distance is proportional to the logarithm of the frequency, and going up one octave in frequency corresponds to moving roughly 3.5 mm. Of the 12,000 or so hair cells in the cochlea most seem to be involved mainly with mechanical issues; about 3500 seem to produce outgoing signals. These are collected by about 30,000 nerve fibers which go down the auditory nerve and after several stops reach the auditory cortex. Different nerve cells seem to have rates of firing which are set up to reflect both sound intensity, and below perhaps 300 Hz, actual amplitude peaks in the sound waveform. Much as in both the visual and tactile systems, there seems to be a fairly direct mapping from position on the cochlea to position in the auditory cortex. In animals such as bats it is known that specific nerve cells respond to particular kinds of frequency changes. But in primates, for example, little is known about exactly what features are extracted in the auditory cortex.

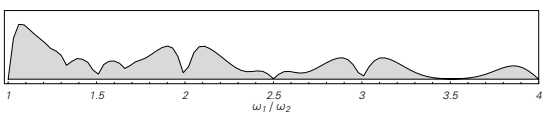
The fact that there are a million nerve fibers going from the eye to the brain, but only about 30,000 going from the ear to the brain means that while it takes several million bits per second to transmit video of acceptable quality, a few tens of thousands of bits are adequate for audio (NTSC television is 5 MHz; audio CDs 22 kHz; telephone 8 kHz). Presumably related is also the fact that it is typically much easier to make realistic sound effects than realistic visual ones.

▪ **Chords.** Two pure tones played together exhibit beats at the difference of their frequencies—a consequence of the fact that

$$Sin[\omega_1 t] + Sin[\omega_2 t] == 2\, Sin[1/2\, (\omega_1 + \omega_2)\, t]\, Cos[1/2\, (\omega_1 - \omega_2)\, t]$$

With $\omega \approx 500$ Hz, one can explicitly hear the time variation of the beats if their frequency is below about 15 Hz, and the result is quite pleasant. But between 15 Hz and about 60 Hz, the sound tends to be rather grating—possibly because this frequency range conflicts with that used for signals in the auditory nerve.

In music it is usually thought that chords consisting of tones with frequencies whose ratios have small denominators (such as 3/2, corresponding to a perfect fifth) yield the most pleasing sounds. The mechanics of the ear imply that if two tones of reasonable amplitude are played together, progressively smaller additional signals will effectively be generated at frequencies $Abs[n_1\, \omega_1 \pm n_2\, \omega_2]$. The picture below shows the extent to which such frequencies tend to be in the range that yield grating effects. The minima at values of ω_2/ω_1 corresponding to rationals with small denominators may explain why such chords seem more pleasing. (See also page 917.)



■ **History.** The notion of musical notes and of concepts such as octaves goes back at least five thousand years. Around 550 BC the Pythagoreans identified various potential connections between numbers and the perception of sounds. And over the course of time a wide range of mathematical and aesthetic principles were suggested. But it was not until the 1800s, particularly with the work of Hermann Helmholtz, that the physical basis for the perception of sound began to be seriously investigated. Work on speech sounds by Alexander Graham Bell and others was related to the development of the telephone in the late 1800s. In the past few decades, with better experiments, particularly on the emission of sound by the ear, and with ideas and analysis from electrical engineers and physicists the basic behavior of at least the cochlea is becoming largely understood.

■ **Sonification.** Sound has occasionally been used as a means of understanding scientific data. In the 1950s and 1960s analog computers (and sometimes digital computers) routinely had sound output. And in the 1970s some discoveries about chaos in differential equations were made using such output. In experimental neuroscience sounds are also routinely used to monitor impulses in nerve cells.

■ **Implementation.** *ListPlay[data]* in *Mathematica* generates sound output by treating the elements of *data* as successive samples in the waveform of the sound, typically with a default sample rate of 8000 Hz.

■ **Time variation.** Many systems discussed in this book produce sounds with distinctive and sometimes pleasing time variation. Particularly dramatic are the concatenation systems discussed on page 913, as well as successive rows in nested patterns such as *Flatten[IntegerDigits[NestList[BitXor[#, 2#] &, 1, 500], 2]]* and sequences based on numbers such as *Flatten[Table[If[GCD[i, j] == 0, 1, 0], {i, 1000}, {j, i}]]* (see page 613). The recursive sequences on page 130 yield sounds reminiscent of many natural systems.

■ **Musical scores.** Instead of taking a sequence to correspond directly to the waveform of a sound, one can consider it to give a musical score in which each element represents a note of a certain frequency, played for some specific short time. (One can avoid clicks by arranging the waveform to cross zero at both the beginning and end of each note.) With this setup my experience is that both repetitive and random sequences tend to seem quite monotonous and dull. But nested sequences I have found can quite often generate rather pleasing tunes. (One can either determine frequencies of notes directly from the values of elements, or, say, from cumulative sums of such values, or from heights in paths like those on page 892.) (See also page 869.)

■ **Recognizing repetition.** The curve of the function $Sin[x] + Sin[\sqrt{2}\, x]$ shown on page 146 looks complicated to the eye. But a sound with a corresponding waveform is recognized by the ear as consisting simply of two pure tones. However, if one uses the function to generate a score—say playing a note at the position of each peak—then no such simplicity can be recognized. And this fact is presumably why musical scores normally have notes only at integer multiples of some fixed time interval.

■ **Sound compression.** Sound compression has in practice mostly been applied to human speech. In typical voice coders (vocoders) 64k bits per second of digital data are obtained by sampling the original sound waveform 8000 times per second, and assigning one of 256 possible levels to each sample. (Since the 1960s, so-called mu-law companding has often been used, in which these levels are distributed exponentially in amplitude.) Encoding only differences between successive samples leads to perhaps a factor of 2 compression. Much more dramatic compression can be achieved by making an explicit model for speech sounds. Most common is to assume that within each phoneme-length chunk of a few tens of milliseconds the vocal tract acts like a linear filter excited either by pure tones or randomness. In so-called linear predictive coding (LPC) optimal parameters are found to make each sound sample be a linear combination of, say, 8 preceding samples. The residue from this procedure is then often fitted to a code book of possible forms, and the result is that intelligible speech can be obtained with as little as 3 kbps of data. Hardware implementations of LPC and related methods have been widespread since before the 1980s; software implementations are now becoming common. Music has in the past rarely been compressed, except insofar as it can be specified by a score. But recently the MP3 format associated with MPEG and largely based on LPC methods has begun to be used for compression of arbitrary sounds, and is increasingly applied to music.

■ **Page 586 · Spectra.** The spectra shown are given by *Abs[Fourier[data]]*, where the symmetrical second half of this list is dropped in the pictures. Also of relevance are intensity or power spectra, obtained as the square of these spectra. These are related to the autocorrelation function according to

```
Fourier[list]^2 ==
  Fourier[ListConvolve[list, list, {1, 1}]]/Sqrt[Length[list]]
```

(See also page 1074.)

■ **Spectra of substitution systems.** Questions that turn out to be related to spectra of substitution systems have arisen in various areas of pure mathematics since the late 1800s. In the 1980s, particularly following discoveries in iterated maps and quasicrystals, studies of such spectra were made in the

context of number theory and dynamical systems theory. Some general principles were proposed, but a great many exceptions were always eventually found.

As suggested by the pictures in the main text, spectra such as (b) and (d) in the limit consist purely of discrete Dirac delta function peaks, while spectra such as (a) and (c) also contain essentially continuous parts. There seems to be no simple criterion for deciding from the rule what type of spectrum will be obtained. (In some cases it works to look at whether the limiting ratio of lengths on successive steps is a Pisot number.) One general result, however, is that all so-called Sturmian sequences `Round[(n + 1) a + b] - Round[n a + b]` with `a` an irrational number must yield discrete spectra. And as discussed on page 903, if `a` is a quadratic irrational, then such sequences can be generated by substitution systems.

For any substitution system the spectrum $\phi[i][t, \omega]$ at step t from initial condition i is given by a linear recurrence relation in terms of the $\phi[j][t-1, \omega]$. With k colors each giving a string of the same length s the recurrence relation is

```
Thread[Map[ϕ[#][t + 1, ω] &, Range[k] - 1] ==
  Apply[Plus, MapIndexed[Exp[i ω (Last[#2] - 1) s^t]
      ϕ[#1][t, ω] &, Range[k] - 1 /. rules, {-1}], {1}]/√s ]
```

Some specific properties of the examples shown include:

(a) *(Thue-Morse sequence)* The spectrum is essentially `Nest[Range[2 Length[#]] Join[#, Reverse[#]] &, {1}, t]`. The main peak is at position 1/3, and in the power spectrum this peak contains half of the total. The generating function for the sequence (with 0 replaced by -1) satisfies $f[z] == (1-z) f[z^2]$, so that $f[z] == \text{Product}[1 - z^{2^n}, \{n, 0, \infty\}]$. (Z transform or generating function methods can be applied directly only for substitution systems with rules such as $\{1 \to \text{list}, 0 \to 1 - \text{list}\}$.) After t steps a continuous approximation to the spectrum is `Product[1 - Exp[2^s i ω], {s, t}]`, which is an example of a type of product studied by Frigyes Riesz in 1918 in connection with questions about the convergence of trigonometric series. It is related to the product of sawtooth functions given by `Product[Abs[Mod[2^s ω, 2, -1]], {s, t}]`. Peaks occur for values of ω such as 1/3 that are not well approximated by numbers of the form $a/2^b$ with small a and b.

(b) *(Fibonacci-related sequence)* This sequence is a Sturmian one. The maximum of the spectrum is at `Fibonacci[t]`. The spectrum is roughly like the markings on a ruler that is recursively divided into `{GoldenRatio, 1}` pieces.

(c) *(Cantor set)* In the limit, no single peak contains a nonzero fraction of the power spectrum. After t steps a continuous approximation to the spectrum is `Product[1 + Exp[3^s 2 i ω], {s, t}]`.

(d) *(Period-doubling sequence)* The spectrum is `(2^# - (-1)^# &)[1 + IntegerExponent[n, 2]]`, almost like the markings on a base 2 ruler.

(See also page 917.)

■ **Flat spectra.** Any impulse sequence `Join[{1}, Table[0, {n}]]` will yield a flat spectrum. With odd n the same turns out to be true for sequences `Exp[2 π i Mod[Range[n]^2, n]/n]`—a fact used in the design of acoustic diffusers (see page 1183). For sequences involving only two distinct integers flat spectra are rare; with ± 1 those equivalent to `{1, 1, 1, -1}` seem to be the only examples. (`{r^2, r s, s^2, -r s}` works for any r and s, as do all lists obtained working modulo $x^n - 1$ from `p[x]/p[1/x]` where `p[x]` is any invertible polynomial.) If one ignores the first component of the spectrum the remainder is flat for a constant sequence, or for a random sequence in the limit of infinite length. It is also flat for maximal length LFSR sequences (see page 1084) and for sequences `JacobiSymbol[Range[0, p - 1], p]` with prime p (see page 870). By adding a suitable constant to each element one can then arrange in such cases for the whole spectrum to be flat. If `Mod[p, 4] == 1` `JacobiSymbol` sequences also satisfy `Fourier[list] == list`. Sequences of 0's and 1's that have the same property are `{1, 0, 1, 0}`, `{1, 0, 0, 1, 0, 0, 1, 0, 0}` or in general `Flatten[Table[{1, Table[0, {n - 1}]}, {n}]]`. If -1 is allowed, additional sequences such as `{0, 1, 0, -1, 0, -1, 0, 1}` are also possible. (See also pages 911.)

■ **Nested vibrations.** With an assembly of springs arranged in a nested pattern simple initial excitations can yield motion that shows nested behavior in time. If the standard methodology of mechanics is followed, and the system is analyzed in terms of normal modes, then the spectrum of possible frequencies can look complicated, just as in the examples on page 586. (Similar considerations apply to the motion of quantum mechanical electrons in nested potentials.)

■ **Page 587 · Random block sequences.** Analytical forms for all but the last spectrum are: 1, $u^2/(1 + 8u^2)$, $1/(1 + 8u^2)$, u^2, $(1 - 4u^2)^2/(1 - 5u^2 + 8u^4)$, $u^2/(1 - 5u^2 + 8u^4)$, $u^2 + 1/36\ \text{DiracDelta}[\omega - 1/3]$, where $u = \text{Cos}[\pi\omega]$, and ω runs from 0 to $1/2$ in each plot. Given a list of blocks such as `{{1, 1}, {0}}` each element of `Flatten[list]` can be thought of as a state in a finite automaton or a Markov process (see page 1084). The transitions between these states have probabilities given by `m[Map[Length, list]]` where

```
m[s_] := With[{q = FoldList[Plus, 0, s]}, ReplacePart[
    RotateRight[IdentityMatrix[Last[q]], {0, 1}], 1/Length[s],
    Flatten[Outer[List, Rest[q], Drop[q, -1] + 1, 1]]]
```

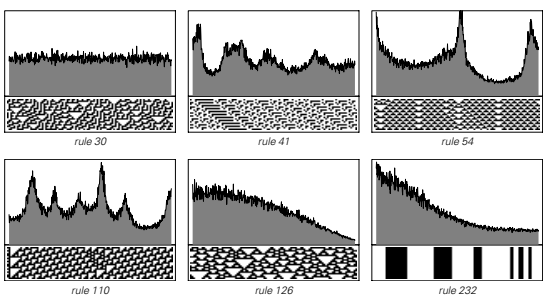
The average spectrum of sequences generated according to these probabilities can be obtained by computing the correlation function for elements a distance r apart

```
ξ[list_, r_] := With[{w = (# - Apply[Plus, #]/Length[#] &)[
      Flatten[list]]}, w . MatrixPower[
      m[Map[Length, list]], r] . w/Length[w]]
```

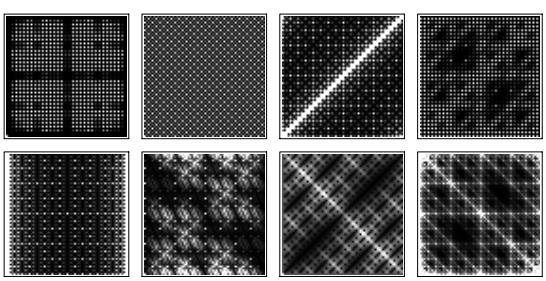
then forming $Sum[\xi[Abs[r]]\,Cos[2\,\pi\,r\,\omega], \{r, -n/2, n/2\}]$ and taking the limit $n \to \infty$. If $\xi[r] = \lambda^r$ then the spectrum is $(1-\lambda^2)/(\lambda^2 - 2\,\lambda\,Cos[2\,\pi\,\omega] + 1) - 1$. For a random walk (see page 977) in which ± 1 occur with equal probability the spectrum is $Csc[\pi\,\omega]^2/2$, or roughly $1/\omega^2$.

The same basic setup also applies to spectra associated with linear filters and ARMA time series processes (see page 1083), in which elements in a sequence are generated from external random noise by forming linear combinations of the noise with definite configurations of elements in the sequence.

▪ **Spectra of cellular automata.** When cellular automata have non-trivial attractors as discussed in Chapter 6 the spectra of sequences obtained at particular steps can exhibit a variety of features, as shown below.



▪ **2D spectra.** The pictures below give the 2D Fourier transforms of the nested patterns shown on page 583.



▪ **Diffraction patterns.** X-ray diffraction patterns give Fourier transforms of the spatial arrangement of atoms in a material. For an ordinary crystal with atoms on a repetitive lattice, the patterns consist of a few isolated peaks. For quasicrystals with generalized Penrose tiling structures the patterns also contain a few large peaks, though as in example (b) on page 586 there are also a hierarchy of smaller peaks present. In general, materials with nested structures do not necessarily yield discrete diffraction patterns. In the early 1990s, experiments were done in which layers a few atoms thick of two different materials were deposited in a Thue-Morse sequence. The resulting object was found to yield X-ray diffraction patterns just like example (a) on page 586.

Statistical Analysis

▪ **History.** Some computations of odds for games of chance were already made in antiquity. Beginning around the 1200s increasingly elaborate results based on the combinatorial enumeration of possibilities were obtained by mystics and mathematicians, with systematically correct methods being developed in the mid-1600s and early 1700s. The idea of making inferences from sampled data arose in the mid-1600s in connection with estimating populations and developing precursors of life insurance. The method of averaging to correct for what were assumed to be random errors of observation began to be used, primarily in astronomy, in the mid-1700s, while least squares fitting and the notion of probability distributions became established around 1800. Probabilistic models based on random variations between individuals began to be used in biology in the mid-1800s, and many of the classical methods now used for statistical analysis were developed in the late 1800s and early 1900s in the context of agricultural research. In physics fundamentally probabilistic models were central to the introduction of statistical mechanics in the late 1800s and quantum mechanics in the early 1900s. Beginning as early as the 1700s, the foundations of statistical analysis have been vigorously debated, with a succession of fairly specific approaches being claimed as the only ones capable of drawing unbiased conclusions from data. The practical use of statistical analysis began to increase rapidly in the 1960s and 1970s, particularly among biological and social scientists, as computers became more widespread. All too often, however, inadequate amounts of data have ended up being subjected to elaborate statistical analyses whose results are then blindly assumed to represent definitive scientific conclusions. In the 1980s, at least in some fields, traditional statistical analysis began to become less popular, being replaced by more direct examination of data presented graphically by computer. In addition, in the 1990s, particularly in the context of consumer electronics

devices, there has been an increasing emphasis on using statistical analysis to make decisions from data, and methods such as fuzzy logic and neural networks have become popular.

■ **Practical statistics.** The vast majority of statistical analysis is in practice done on continuous numerical data. And with surprising regularity it is assumed that random variations in such data follow a Gaussian distribution (see page 976). But while this may sometimes be true—perhaps as a consequence of the Central Limit Theorem—it is rarely checked, making it likely that many detailed inferences are wrong. So-called robust statistics uses for example medians rather than means as an attempt to downplay outlying data that does not follow a Gaussian distribution.

Classical statistical analysis mostly involves trying to use data to estimate parameters in specific probabilistic models. Non-parametric statistics and related methods often claim to derive conclusions without assuming particular models for data. But insofar as a conclusion relies on extrapolation beyond actual measured data it must inevitably in some way use a model for data that has not been measured.

■ **Time series.** Sequences of continuous numerical data are often known as time series, and starting in the 1960s standard models for them have consisted of linear recurrence relations or linear differential equations with random noise continually being added. The linearity of such models has allowed efficient methods for estimating their parameters to be developed, and these are widely used, under slightly different names, in control engineering and in business analysis. In recent years nonlinear models have also sometimes been considered, but typically their parameters are very difficult to estimate reliably. As discussed on page 919 it was already realized in the 1970s that even without external random noise nonlinear models could produce time series with seemingly random features. But confusion about the importance of sensitivity to initial conditions caused the kind of discoveries made in this book to be missed.

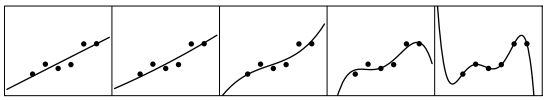
■ **Page 588 · Origin of probabilities.** Probabilities are normally assumed to enter for at least two reasons: (a) because of random variation between individuals, and (b) because of random errors in measurement. (a) is particularly common in the biological and social sciences; (b) in the physical sciences. In physics effects of statistical mechanics and quantum mechanics are also assumed to introduce probabilities. Probabilistic models for abstract mathematical systems have in the past been rare, though the results about randomness in this book may make them more common in the future.

■ **Probabilistic models.** A probabilistic model must associate with every sequence a probability that is a number between 0 and 1. This can be done either by giving an explicit procedure for taking sequences and finding probabilities, or by defining a process in which sequences are generated with appropriate probabilities. A typical example of the first approach is the Ising model for spin systems in which relative probabilities of sequences are found by multiplying together the results of applying a simple function to blocks of nearby elements in the sequence. Monte Carlo methods and probabilistic cellular automata provide examples of the second approach.

■ **Page 588 · Binomial distribution.** If black squares appear independently with probability p then the probability that m squares out of n are black is $\mathit{Binomial}[n, m]\, p^m\, (1-p)^{n-m}$.

■ **Page 589 · Estimation of parameters.** One way to estimate parameters in simple probabilistic models is to compute the mean and other moments of the data and then to work out what values of the parameters will reproduce these. More general is the maximum likelihood method in which one finds the values of the parameters which maximize the probability of generating the observed data from the model. (Least squares fits do this for models in which the data exhibits independent Gaussian variations.) Various modifications can be made involving for example weighting with a risk function before maximizing. If one starts with a priori probability distributions for all parameters, then Bayes's Theorem on conditional probabilities allows one to avoid the arbitrariness of methods such as maximum likelihood and explicitly to work out from the observed data what the probability is for each possible choice of parameters in the model. It is rare in practice, however, to be able to get convincing *a priori* probability distributions, although when there are physical or other reasons to expect entropy to be maximized the so-called maximum entropy method may be useful.

■ **Complexity of models.** The pictures at the top of the next page show least squares fits (found using *Fit* in *Mathematica*) to polynomials with progressively higher degrees and therefore progressively more parameters. Which fit should be considered best in any particular case must ultimately depend on external considerations. But since the 1980s there have been attempts to find general criteria, typically based on maximizing quantities such as $-\mathit{Log}[p] - d$ (the Akaike information criterion), where p is the probability that the observed data would be generated from a given model ($-\mathit{Log}[p]$ is proportional to variance in a least squares fit), and d is the number of parameters in the model.



■ **Page 590 · Markov processes.** The networks in the main text can be viewed as representing finite automata (see page 957) with probabilities associated with transitions between nodes or states. Given a vector of probabilities to be in each state, the evolution of the system corresponds to multiplication by the matrix of probabilities for each transition. (Compare the calculation of properties of substitution systems on page 890.) Markov processes first arose in the early 1900s and have been widely studied since the 1950s. In their first uses as models it was typically assumed that each state transition could explicitly be observed. But by the 1980s hidden Markov models were being studied, in which only some of the states or transitions could be distinguished by outside observations. Practical applications were made in speech understanding and text compression. And in the late 1980s, building on work of mine from 1984 (described on page 276), James Crutchfield made a study of such models in which he defined the complexity of a model to be equal to $-p\,Log[p]$ summed over all connections in the network. He argued that the best scientific model is one that minimizes this complexity—which with probabilities 0 and 1 is equivalent to minimizing the number of nodes in the network.

■ **Non-local processes.** It follows from the fact that any path in a finite network must always eventually return to a node where it has been before that any Markov process must be fundamentally local, in the sense that the probabilities it implies for what happens at a given point in a sequence must be independent of those for points sufficiently far away. But probabilistic models based on other underlying systems can yield sequences with long-range correlations. As an example, probabilistic neighbor-independent substitution systems can yield sequences with hierarchical structures that have approximate nesting. And since the mid-1990s such systems (usually characterized as random trees or random context-free languages) have sometimes been used in analyzing data that is expected to have grammatical structure of some kind.

■ **Page 594 · Block frequencies.** In any repetitive sequence the number of distinct blocks of length m must become constant with m for sufficiently large m. In a nested sequence the number must always continue increasing roughly linearly, and must be greater than m for every m. (The differences of successive numbers themselves form a nested sequence.) If exactly $m+1$ distinct blocks occur for every m, then the sequence must be of the so-called Sturmian type discussed on page 916, and the n^{th} element must be given by $Round[(n+1)a+b]-Round[na+b]$, where a is an irrational number. Up to limited m nested sequences can contain all k^m possible blocks, and can do so with asymptotically equal frequencies. Pictures (b), (c) and (d) show the simplest cases where this occurs (for length 3 $\{1\to\{1,1,1,0,0,0\},0\to\{1,0\}\}$ also works). Linear feedback shift registers of the type used in picture (e) are discussed below. Concatenation sequences of the type used in picture (f) are discussed on page 913. In both cases equal frequencies of blocks are obtained only for sequences of length exactly 2^i.

■ **LFSR sequences.** Often referred to as pseudonoise or PN sequences, maximal length linear feedback shift register sequences have repetition period 2^n-1 and are generated by shift registers that go through all their possible states except the one consisting of all 0's, as discussed on page 974. Blocks in such sequences obtained from $Partition[list, n, 1]$ must all be distinct since they correspond to successive complete states of the shift register. This means that every block with length up to n (except all 0's) must occur with equal frequency. (Note that only a small fraction of all possible sequences with this property can be generated by LFSRs.) The regularity of PN sequences is revealed by looking at the autocorrelation $RotateLeft[(-1)^{list}, m]\,.\,(-1)^{list}$. This quantity is -1 for all nonzero m for PN sequences (so that all but the first component in $Abs[Fourier[(-1)^{list}]]^2$ are equal), but has mean 0 for truly random sequences. (Related sequences can be generated from $RealDigits[1/p, 2]$ as discussed on page 912.)

■ **Entropy estimates.** Fitting the number of distinct blocks of length b to the form k^{hb} for large b the quantity h gives the so-called topological entropy of the system. The so-called measure entropy is given as discussed on page 959 by the limit of $-Sum[p_i\,Log[k, p_i], \{i, k^b\}]/b$ where the p_i are the probabilities for the blocks. Actually getting accurate estimates of such entropies is however often rather difficult, and typically upper bounds are ultimately all that can realistically be given. Note also that as discussed in the main text having maximal entropy does not by any means imply perfect randomness.

■ **Tests of randomness.** Statistical analysis has in practice been much more concerned with finding regularities in data than in testing for randomness. But over the course of the past century a variety of tests of randomness have been proposed, especially in the context of games of chance and their government regulation. Most often the tests are applied not directly to sequences of 0's and 1's, but instead say to numbers obtained from blocks of 8 elements. A typical collection of tests described by Donald Knuth in 1968 includes: (1) frequency or equidistribution test (possible

elements should occur with equal frequency); (2) serial test (pairs of elements should be equally likely to be in descending and ascending order); (3) gap test (runs of elements all greater or less than some fixed value should have lengths that follow a binomial distribution); (4) poker test (blocks corresponding to possible poker hands should occur with appropriate frequencies); (5) coupon collector's test (runs before complete sets of values are found should have lengths that follow a definite distribution); (6) permutation test (in blocks of elements possible orderings of values should occur equally often); (7) runs up test (runs of monotonically increasing elements should have lengths that follow a definite distribution); (8) maximum-of-t test (maximum values in blocks of elements should follow a power-law distribution). With appropriate values of parameters, these tests in practice tend to be at least somewhat independent, although in principle, if sufficient data were available, they could all be subsumed into basic block frequency and run-length tests. Of the sequences on page 594, (a) through (d) as well as (f) fail every single one of the tests, (e) fails only the serial test, while (g) and (h) pass all the tests. (Failure is defined as a value that is as large or small as that obtained from the data occurring below a specified probability in the set of all possible sequences.) Widespread use of tests like these on pseudorandom generators (see page 974) began in the late 1970s, with discoveries of defects in common generators being announced every few years.

In the 1980s simulations in physics had begun to use pseudorandom generators to produce sequences with billions of elements, and by the late 1980s evidence had developed that a few common generators gave incorrect results in such cases as phase transition properties of the 3D Ising model and shapes of diffusion-limited aggregates. (These difficulties provided yet more support for my contention that models with intrinsic randomness are more reliable than those with external randomness.) In the 1990s various idealizations of physics simulations—based on random walks, correlation functions, localization of eigenstates, and so on—were used as tests of pseudorandom generators. These tests mostly seem simpler than those shown on page 597 obtained by running a cellular automaton rule on the data.

Over the years, essentially every proposed statistical test of randomness has been applied to the center column of rule 30. And occasionally people have told me that their tests have found deviations from randomness. But in every single case further investigation showed that the results were somehow incorrect. So as of now, the center column of rule 30 appears to pass every single proposed statistical test of randomness.

■ **Difference tables.** See page 1091.

■ **Randomized algorithms.** Whether a randomized algorithm gives correct answers can be viewed as a test of randomness for whatever supposedly random sequence is provided to it. But in most practical cases such tests are not particularly stringent; linear congruential generators, for example, almost always pass. (There are perhaps exceptions in VLSI testing.) And this is basically why it has so often proved possible to replace randomized algorithms by deterministic ones that are at least as efficient (see page 1192). An example is Monte Carlo integration, where what ultimately matters is uniform sampling of the integrand—which can usually be achieved better by quasi-random irrational number multiple (see page 903) or digit reversal (see page 905) sequences than by sequences one might consider more random.

Cryptography and Cryptanalysis

■ **History.** Cryptography has been in use since antiquity, and has been a decisive factor in a remarkably large number of military and other campaigns. Typical of early systems was the substitution cipher of Julius Caesar, in which every letter was cyclically shifted in the alphabet by three positions, with A being replaced by D, B by E, and so on. Systems based on more arbitrary substitutions were in use by the 1300s. And while methods for their cryptanalysis were developed in the 1400s, such systems continued to see occasional serious use until the early 1900s. Ciphers of the type shown on page 599 were introduced in the 1500s, notably by Blaise de Vigenère; systematic methods for their cryptanalysis were developed in the mid-1800s and early 1900s. By the mid-1800s, however, codes based on books of translations for whole phrases were much more common than ciphers, probably because more sophisticated algorithms for ciphers were difficult to implement by hand. But in the 1920s electromechanical technology led to the development of rotor machines, in which an encrypting sequence with an extremely long period was generated by rotating a sequence of noncommensurate rotors. A notable achievement of cryptanalysis was the 1940 breaking of the German Enigma rotor machine using a mixture of statistical analysis and automatic enumeration of keys. Starting in the 1950s, electronic devices were the primary ones used for cryptography. Linear feedback shift registers and perhaps nonlinear ones seem to have been common, though little is publicly known about military cryptographic systems after World War II. In 1977 the U.S. government introduced the DES data encryption standard, and in the 1980s this became the dominant force in the growing field of commercial cryptography. DES takes 64-bit

blocks of data and a 56-bit key, and applies 16 rounds of substitutions and permutations. The S-box that implements each substitution works much like a single step of a cellular automaton. No fast method of cryptanalysis for DES is publicly known, although by now for a single DES system an exhaustive search of keys has become feasible. Two major changes occurred in cryptography in the 1980s. First, cryptographic systems routinely began to be implemented in software rather than in special-purpose hardware, and thus became much more widely available. And second, following the introduction of public-key cryptography in 1975, the idea emerged of basing cryptography not on systems with complicated and seemingly arbitrary construction, but instead on systems derived from well-known mathematical problems. Initially several different problems were considered, but after a while the only ones to survive were those such as the RSA system discussed below based essentially on the problem of factoring integers. Present-day publicly available cryptographic systems are almost all based on variants of either DES (such as the IDEA system of PGP), linear feedback shift registers or RSA. My cellular automaton cryptographic system is one of the very few fundamentally different systems to have been introduced in recent years.

■ **Basic theory.** As was recognized in the 1920s the only way to make a completely secure cryptographic system is to use a so-called one-time pad and to have a key that is as long as the message, and is chosen completely at random separately for each message. As soon as there are a limited number of possible keys then in principle one can always try each of them in turn, looking in each case to see whether they imply an original message that is meaningful in the language in which the message is written. And as Claude Shannon argued in the 1940s, the length of message needed to be reasonably certain that only one key will satisfy this criterion is equal to the length of the key divided by the redundancy of the language in which the message is written—equal to about 0.5 for English (see below).

In a cryptographic system with keys of length n there will typically be a total of k^n possible keys. If one guesses a key it will normally take a time polynomial in n to check whether the key is correct, and thus the problem of cryptanalysis is in the class known in theoretical computer science as NP or non-deterministic polynomial time (see page 1142). It is suspected but not established that there exist at least some problems in NP that cannot be solved in polynomial time, potentially indicating that for an appropriate system it might be impossible to do cryptanalysis in any time polynomial in n. (See page 1089.)

■ **Text.** As the picture below illustrates, English text typically remains intelligible until about half its characters have been deleted, indicating that it has a redundancy of around 0.5. Most other languages have slightly higher redundancies, making documents in those languages slightly longer than their counterparts in English.

```
About half the letters in typical English text are redundant.
About half the letter- in typical Eng--sh text are redun-ant.
Abou- half the -etter- in ty-ical Eng--sh text are redun--nt.
Abou- half the -e-t--- i- ty-ical Eng--sh text are redun--nt.
Abou- half t-e -e-t--- -- ty-ical Eng--sh text ar- red-n--nt.
Abou- h-l- t-e -e----- -- ty-ical Eng--sh text ar- r-d-n--nt.
Abou- h-l- t-- -e----- -- ty-ical -ng--sh tex- ar- --d-n--nt.
Abou- h--- --- -e----- -- ty-ica- -ng---h tex- ar- --d-n--nt.
Abou- h--- --- -e----- -- ty---a- -ng---h te-- -r- --d-n--nt.
A-ou- h--- --- -e----- -- ty---a- -ng---- te-- -r- --d----n-.
--ou- ---- --- -e----- -- ty---a- -n----- te-- -r- -------n-.
--o-- ---- --- -e----- -- ty---a- ------- -e-- -r- ---------.
----- ---- --- -e----- -- t------ ------- -e-- --- ---------.
```

Redundancy can in principle be estimated by breaking text into blocks of length b, then looking for the limit of the entropy as $b \to \infty$ (see page 1084). Statistically uniform samples of text do not in practice, however, tend to be large enough to allow more than about $b = 6$ to be reached, and the presence of correlations (even though exponentially damped) between far-separated letters means that computed entropies usually decrease continually with b, making it difficult to estimate their limit (see page 1084). Note that particularly in computer languages higher redundancy is found if one takes account of grammatical structure.

■ **Page 599 · Cryptanalysis.** The so-called Vigenère cipher was thought for several centuries to be unbreakable. The idea of looking for repeats was introduced by Friedrich Kasiski in 1863. A statistical approach based on the fact that frequencies tend to be closer to uniform for longer keys was introduced by William Friedman in the 1920s. The methods described in the main text are fairly characteristic of the mixture between generality and detail that is typical in practical cryptanalysis.

■ **Page 600 · Linear feedback shift registers.** See notes on pages 974 and 1084. LFSR sequences are widely used in radio technology, particularly in the context of spread spectrum applications. Their purpose is usually to provide a way to distinguish or synchronize signals, and sometimes to provide a level of cryptographic security. In CDMA technology for cellular telephones, for example, data is overlaid on LFSR sequences, and sequences other than the one intended for a particular receiver seem like noise which can be ignored. As another example, the Global Positioning System (GPS) works by having 24 satellites each transmit maximal length sequences from different length 10 LFSRs. Position is deduced from the arrival times of signals, as determined by the relative phases of the LFSR sequences received. (GPS P-code apparently uses much longer LFSR sequences and repeats only every 267 days. Before May 2000 it was used to add unpredictable timing errors to ordinary GPS signals.)

■ **LFSR cryptanalysis.** Given a sequence obtained from a length n LFSR (see page 975)

```
Nest[Mod[Append[#, Take[#, -n] . vec], 2] &, list, t]
```

the vector of taps *vec* can be deduced from

```
LinearSolve[Table[Take[seq, {i, i + n - 1}], {i, n}],
  Take[seq, {n + 1, 2 n}], Modulus → 2]
```

(An iterative algorithm in n taking about n^2 rather than n^3 steps was given by Elwyn Berlekamp and James Massey in 1968.) The same basic approach can be used to deduce the rule for an additive cellular automaton from vertical sequences.

■ **Page 603 · Rule 30 cryptography.** Rule 30 is known to have many of the properties desirable for practical cryptography. It does not repeat with any short period or show any obvious structure for almost all keys. Small changes in keys typically leads to large changes in the encrypting sequence. The Boolean expressions which determine the encrypting sequence from the key rapidly become highly complex (see page 618). And furthermore the system can be implemented very efficiently, particularly in parallel hardware.

I originally studied rule 30 in the context of basic science, but I soon realized that it could serve as the basis for practical random sequence generation and cryptography, and I analyzed this extensively in 1985. (Most but not all of the results from my original paper are included in this book, together with various new results.) In 1985 and soon thereafter a number of people (notably Richard and Carl Feynman) tried to cryptanalyze rule 30, but without success. From the beginning, computations of spacetime entropies for rule 30 (see page 960) gave indications that for strong cryptography one should not sample all cells in a column, and in 1991 Willi Meier and Othmar Staffelbach described essentially the explicit cryptanalysis approach shown on page 601. Rule 30 has been widely used for random sequence generation, but for a variety of reasons I have not in the past much emphasized its applications in cryptography.

■ **Properties of rule 30.** Rule 30 can be written in the form $p \veebar (q \vee r)$ (see page 869) and thus exhibits a kind of one-sided additivity on the left. This leads to some features that are desirable for cryptography (such as long repetition periods) and to some that are not (such as the sideways evolution of page 601). It implies that every block of length m that occurs at a particular step has exactly 4 immediate predecessor blocks of length $m+2$ (see page 960). It also implies that all 2^t possible single columns of t cells can be generated from some initial condition. Not all 4^t pairs of adjacent columns can occur, however. There seems to be no simple characterization, say in terms of paths through networks, of which can, but for successive t the total numbers are

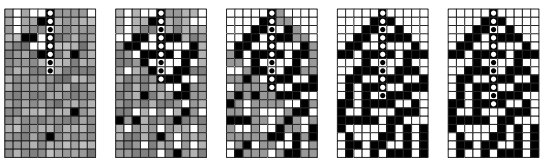
```
{4, 12, 32, 80, 200, 496, 1208, 2916, 6964, 16476, 38616,
  89844, 207544, 476596, 1089000, 2477236, 5615036}
```

or roughly 2.25^t.

Given two complete adjacent columns page 601 shows how all columns any distance to the left can be found. It turns out that this can be done even if the right-hand one of the two adjacent columns is not complete. So for example whenever there is a black cell in the left column it is irrelevant what appears in the right column. Note that the configuration of relevant cells can be repetitive only if the initial conditions were repetitive (see page 871).

In a cellular automaton of limited size n, any column must eventually repeat. There could be 2^n distinct possible columns; in practice, for successive n there are $\{2, 3, 7, 14, 30, 60, 101, 245, 497, 972, 1997, 3997\}$—within 2% of 2^n already for $n = 12$. This means that for the initial conditions to be determined uniquely, the number of cells that must be given in a column is almost exactly n, as illustrated in the pictures below. Many distinct columns correspond to starting at different points on a single cycle of states. The length of the longest cycle grows roughly like $2^{0.63\,n}$ (see page 260). The complete cycle structure is illustrated on page 962. Most of the 2^n possible states have unique predecessors; for large n, about $2^{0.76\,n}$ or $\text{Root}[\#^3 - \#^2 - 2 \&, 1]^n$ instead have 0 or 2 predecessors. The predecessors of a given state can be found from

```
Cases[Map[Fold[Prepend[#1, If[#2 == 1 ⊻
      Take[#1, 2] == {0, 0}, 0, 1]] &, #, Reverse[list]] &, {{0,
  0}, {0, 1}, {1, 0}, {1, 1}}], {a_, b_, c___, a_, b_} → {b, c, a}]
```

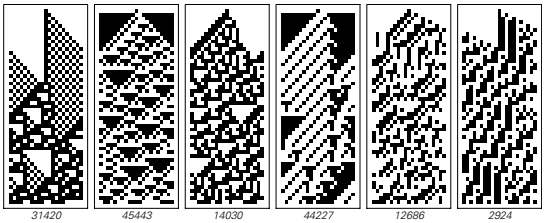


■ **Directional sampling.** One can consider sampling cells not in a vertical column but on lines at any angle. In a rule 30 system of infinite size, it turns out that at 45° clockwise from vertical all possible sequences can occur on any two adjacent lines, probably making cryptanalysis more difficult in this case. (Note that directional sampling is always equivalent to looking at a vertical column in the evolution of a cellular automaton whose basic rule has been composed with an appropriate shift rule.)

■ **Alternative rules.** Among elementary rules, rule 45 is the only plausible alternative to rule 30. It usually yields longer

repetition periods (see page 260), but shows slightly slower responses to changes in the key. (Changes expand about 1.24 cells per step in rule 30, and about 1.17 in rule 45.) Rule 45 shares with rule 30 the property of one-sided additivity. With the occasional exception of the additive rule 60, elementary rules not equivalent to 30 or 45 tend to exhibit vastly shorter repetition periods. (The completely non-additive rule with largest typical repetition period is rule 110.) (See page 951.)

If one considers rules that depend on 4 rather than 3 cells, then the results turn out to be surprisingly similar: out of all 65536 possible such rules the ones with longest periods essentially always seem to be variants of rules 45, 30 or 60. In a region of size 15, for example, the longest period is 20460, and this is achieved by rule 13251, which is just rule 45 applied to the first three cells in the neighborhood. (Rule 45 itself has period 6820 in this case.) After a few rules with long periods, the periods obtained drop off rapidly. (In general the number of rules with a given period seems to decrease roughly exponentially with period.) For size 15, the 33 rules with the longest periods are all additive with respect to one position. The pictures below show the first rules that are not additive with respect to any position.



Among the 4,294,967,296 $r = 2$ rules which depend on 5 cells, there are again just a few that give long periods, but now only a small fraction of these seem directly related to rules 45 and 30, and perhaps half are not additive with respect to any position. The pictures below show the rules with longest periods for size 15; these same rules also yield the longest periods for many other sizes. The first two are additive with respect to one position, but do not appear to be directly related to rules 45 or 30; the last two are not additive with respect to any position. Formulas for the rules are respectively:

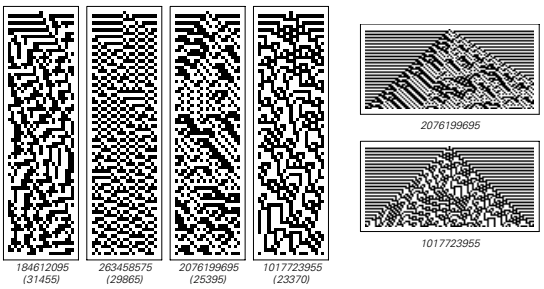
$p \veebar (\neg q \vee r \vee s \wedge \neg t)$

$r \veebar (\neg p \vee q \vee s \wedge \neg t)$

$u = \neg p \wedge \neg q \vee q \wedge t;\ \neg r \wedge u \vee q \wedge \neg s \wedge (p \vee \neg r) \vee r \wedge s \wedge \neg u$

$s \wedge (q \wedge \neg r \vee p \wedge \neg q \wedge t) \vee \neg (s \vee (p \vee q) \wedge (r \veebar (q \vee t)))$

Note that for size 15 the maximum possible period is 32730 (see page 950).



■ **Nonlinear feedback shift registers.** Linear feedback shift registers of the kind discussed on page 974 can be generalized to allow any function f (note the slight analogy with cyclic tag systems):

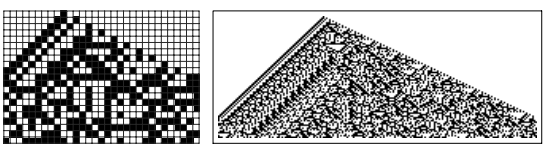
```
NLFSRStep[f_, taps_, list_] :=
  Append[Rest[list], f[list[[taps]]]]
```

With the choice `f = IntegerDigits[s, 2, 8][[8 - # . {4, 2, 1}]] &` and `taps = {1, 2, 3}` this is essentially a rule s elementary cellular automaton. With a list of length n, `Nest[NLFSRStep[f, taps, #] &, list, n]` gives one step in the evolution of the cellular automaton in a register of width n, with a certain kind of spiral boundary condition. The case analogous to rule 30 yields some of the longest repetition periods—usually remarkably close to the absolute maximum of $2^n - 1$ (for $n = 21$ the result is 1999864, 95% of the maximum).

Nonlinear feedback shift registers were apparently studied in the context of military cryptography in the 1950s, but very little about them has made its way into the open literature (see page 878). An empirical investigation of repetition periods in such systems was made by Solomon Golomb in 1959. The main conclusion drawn from extensive data was that nothing like the linear theory applies. One set of computations concerned functions

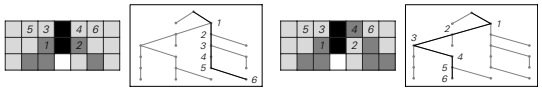
```
f[{w_, x_, y_, z_}] := Mod[w + y + z + x y + x z + y z, 2]
```

(apparently chosen to have balance between 0's and 1's that would minimize correlations). Tap positions `{1, 2, 3, 4}` were among those studied, but nothing like the pictures below were apparently ever explicitly generated—and nearly three decades passed before I noticed the remarkable behavior of the rule 30 cellular automaton.

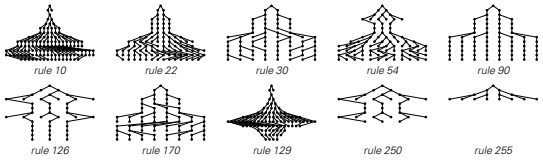


Sequences of states in any shift register must correspond to paths through a network of the kind shown on page 941. And as noted by Nicolaas de Bruijn in 1946 there are $2^{2^{n-1}-n}$ such paths with length 2^n, and thus this number of functions f out of the 2^{2^n} possible must yield sequences of maximal length. (For k colors, the number of paths is $k!^{k^{n-1}}/k^n$.)

■ **Backtracking.** If one wants to find out which of the 2^n possible initial conditions of width n evolve to yield a specific column of colors in a system like an elementary cellular automaton one can usually do somewhat better than just testing all possibilities. The picture below illustrates a typical approach, applied to 3 steps of rule 30. The idea is successively to look at each numbered cell, and to make a tree of possibilities representing what happens if one tries to fill in each possible color for each cell. A branch in the resulting tree continues only if it corresponds to a configuration of cell colors whose evolution is consistent with the specified column of colors.

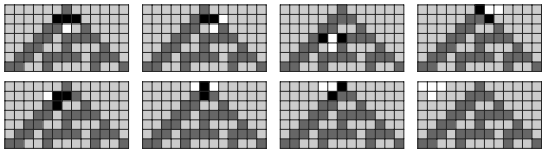


The picture below shows trees obtained for the column ■□■ in various elementary cellular automata. In cases like rules 250 and 254 no initial condition gives the specified column, so all branches eventually die out. In class 2 examples like rule 10 many intermediate configurations are possible. Rules like 90 and to some extent 30 that allow sideways evolution yield comparatively simple trees.



If one wants to find just a single initial condition that works then one can set up a recursive algorithm that in effect does a depth-first traversal of the tree. No doubt in many cases the number of nodes that have to be visited eventually increases like 2^t, but many branches usually die off quickly, greatly reducing the typical effort required in practice.

■ **Deducing cellular automaton rules.** Given a complete cellular automaton pattern it is easy to deduce the rule which produced it just by identifying examples of places where each element in the rule was used, as in the picture at the top of the next column. Given an incomplete pattern, deducing the rule in effect requires solving Boolean equations.



■ **Linear congruential generators.** Cryptanalysis of linear congruential generators is fairly straightforward. Given only an output list *NestList[Mod[a#, m] &, x, n]* parameters *{a, m}* that generate the list can be found for sufficiently large *n* from

```
With[{α = Apply[#2 . Rest[list]/#1 &, Apply[
      ExtendedGCD, Drop[list, -1]]]}, ({Mod[α, #], #} &)[
  Fold[GCD[#1, If[#1 == 0, #2, Mod[#2, #1]]] &, 0,
   ListCorrelate[{α, -1}, list]]]]
```

With slightly more effort both *x* and *{a, m}* can be found just from *First[IntegerDigits[list, 2, p]]*.

■ **Digit sequence encryption.** One can consider using as encrypting sequences the digit sequences of numbers obtained from standard mathematical functions. As discussed on page 139 such digit sequences often seem locally very random. But in many cases one can immediately tell how a sequence was made just by globally applying appropriate mathematical functions. Thus, for example, given the digit sequence of $\sqrt{s}$ one can retrieve the key s just by squaring the number obtained from early digits in the sequence. Whenever a number x is known to satisfy *Sum[a[i] f[i][x], {i, n}] == 0* with fixed *f[i]* one can take the early digits of x and use *LatticeReduce* to find integer solutions for the *a[i]*. With *f[i_] = #^i &* this method allows algebraic numbers to be recognized. If no linear equation is satisfied by any combination of known functions of x, however, the method fails, and it seems quite likely that in such cases secure encrypting sequences can be generated, albeit less efficiently than with systems like cellular automata.

■ **Problem-based cryptography.** Particularly following the work of Whitfield Diffie and Martin Hellman in 1976 it became popular to consider cryptography systems based on mathematical problems that are easy to state but have been found difficult to solve. It was at first hoped that the problems could be NP-complete ones, which are universal in the sense that their solution can be used to provide a solution to any problem in the class NP (see page 1086). To date, however, no system has been devised whose cryptanalysis is known to be NP-complete. Indeed, essentially the only problem on which cryptography systems have so far successfully been based is factoring of integers (see below). And while this problem has resisted a fair number of

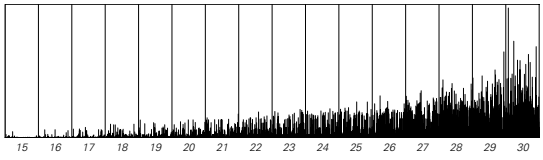
attempts at solution, it is not known to be NP-complete (and indeed its ability to be solved in polynomial time on a formal quantum computer may suggest that it is not).

My cellular automaton cryptography system follows the principle of being based on a problem that is easy to state. And indeed the general problem of finding initial conditions for a cellular automaton is NP-complete (see page 767). But the problem is not known to be NP-complete for the specific case of, say, rule 30. Significantly less work has been done on the problem of finding initial conditions for rule 30 than on the problem of factoring integers. But the greater simplicity of rule 30 might make one already have almost as much confidence in the difficulty of solving this problem as of factoring integers.

■ **Factoring integers.** The difficulty of factoring is presumably related to the irregularity of the pattern of divisors shown on page 909. One approach to factoring a number n is just to try dividing it by each of the numbers up to $\sqrt{n}$. A sequence of much faster methods have however been developed over the past few decades, one simple example that works for most n being the so-called rho method of John Pollard (compare the quadratic residue sequences discussed below):

```
Module[{f = Mod[#^2 + 1, n] &, a = 2, b = 5, c},
  While[(c = GCD[n, a - b]) == 1, {a, b} = {f[a], f[f[b]]}]; c]
```

Most existing methods depend on facts in number theory that are fairly easy to state, though implementing them for maximum efficiency tends to lead to complex programs. Typical running times for *FactorInteger[n]* in *Mathematica* 4 are shown below for the first 1000 numbers with each of 15 through 30 digits. Different current methods asymptotically require slightly different numbers of steps—but all typically at least *Exp[Sqrt[Log[n]]]*. Nevertheless, to test whether a number is prime (*PrimeQ*) it is known that only a few more than *Log[n]* steps suffice.

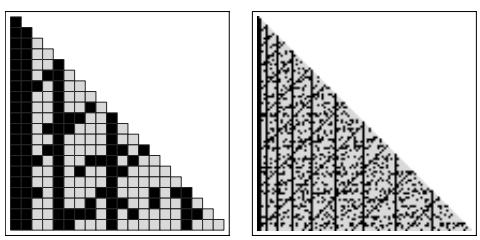


■ **RSA cryptography.** Widely used in practice, the idea is to encode messages using a public key specified by a number n, but to make it so that to decode the messages requires a private key based on the factors of n. An element m in a message is encoded as *c = PowerMod[m, d, n]*. It can then be decoded as *PowerMod[c, e, n]*, where *e = PowerMod[d, -1, EulerPhi[n]]*. But to find *EulerPhi[n]* (see page 1093) is equivalent in difficulty to finding the factors of n.

■ **Quadratic residue sequences.** As an outgrowth of ideas related to RSA cryptography it was shown in 1982 by Lenore Blum, Manuel Blum and Michael Shub that the sequence

```
Mod[NestList[Mod[#^2, m] &, x0, n], 2]
```

discussed on page 975 has the property that if $m = p\,q$ with p and q primes (congruent to 3 modulo 4) then any systematic regularities detected in the sequence can eventually be used to discover factors of m. What is behind this is that each of the numbers in the basic sequence here must be a so-called quadratic residue of the form *Mod[v^2, m]*, and given any such quadratic residue x the expression *GCD[x + Mod[x^2, m], m]* turns out always to be a factor of m—and at least sometimes a non-trivial one. So if one could reconstruct sufficiently many complete numbers x from the sequence of *Mod[x, 2]* values then this would provide a way to factor m (compare the Pollard rho method above). But in practice it is difficult to do this, because without knowing the factors of m one cannot even readily tell whether a given x is a quadratic residue modulo m. The pictures below show as black squares all the quadratic residues for each successive m going down the page (the ordinary squares 1, 4, 9, 16, … show up as vertical black stripes). If m is a prime p, then the simple tests *JacobiSymbol[x, p] == 1* (see page 1081) or *Mod[x^((p-1)/2), p] == 1* determine whether x is a quadratic residue. But with $m = p\,q$, one has to factor m and find p and q in order to carry out similar tests. The condition *Mod[p, 4] == Mod[q, 4] == 3* ensures that only one of the solutions $+v$ and $-v$ to *x == Mod[v^2, m]* is ever a quadratic residue, with the result that the iterated mapping $x \to$ *Mod[x^2, m]* always has a unique inverse. But unlike in a cellular automaton even given a complete x (the analog of a complete cellular automaton state) it is difficult to invert the mapping and solve for the x on the previous step.

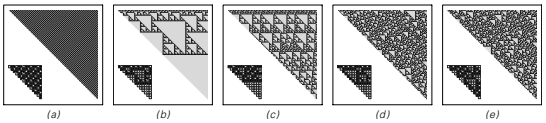


Traditional Mathematics and Mathematical Formulas

■ **Practical empirical mathematics.** In looking for formulas to describe behavior seen in this book I have in practice typically taken associated sequences of numbers and then

tested whether obvious regularities are revealed by combinations of such operations as: computing successive differences (see note below), computing running totals, looking for repeated blocks, picking out running maxima, picking out numbers with particular modular residues, and looking at positions of particular values, and at the forms of the digit sequences of these positions.

▪ **Difference tables and polynomials.** A common mathematical approach to analyzing sequences is to form a difference table by repeatedly evaluating *d[list_] := Drop[list, 1] - Drop[list, -1]*. If the elements of *list* correspond to values of a polynomial of degree *n* at successive integers, then *Nest[d, list, n + 1]* will contain only zeros. If the differences are computed modulo *k* then the difference table corresponds essentially to the evolution of an additive cellular automaton (see page 597). The pictures below show the results with *k = 2* (rule 60) for (a) *Fibonacci[n]*, (b) Thue-Morse sequence, (c) Fibonacci substitution system, (d) *(Prime[n] - 1)/2*, (e) digits of π. (See also page 956.)



▪ **Page 607 · Implementation.** The color of a cell at position *{x, y}* in the pattern shown is given by *Extract[{{1, 0, 1}, {0, 1, 0}}, Mod[{y, x}, {2, 3}] + 1]*.

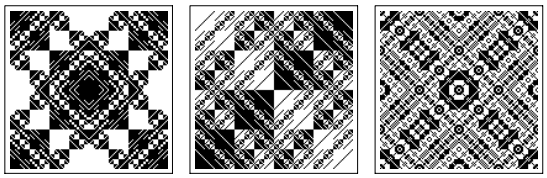
▪ **Page 608 · Nested patterns and numbers.** See page 931.

▪ **Page 609 · Implementation.** Given the rules for a substitution system in the form used on page 931 a finite automaton (as on page 957) which yields the color of each cell from the digit sequences of its position is

```
Map[Flatten[MapIndexed[#2 - 1 → Position[rules, #1 → _][[
    1, 1]] &, Last[#], {-1}]] &, rules]
```

This works in any number of dimensions so long as each replacement yields a block of the same cuboidal form.

▪ **Arbitrary digit operations.** If the operation on digit sequences that determines whether a square will be black can be performed by a finite automaton (see page 957) then the pattern generated must always be either repetitive or nested. The pictures below show examples with more general operations. Picture (a) in effect shows which words in a simple context-free language of parenthesis matching (see page 939) are syntactically correct. Scanning the digit sequences from the left, one starts with 0 open parentheses, then adds 1 whenever corresponding digits in the *x* and *y* coordinates differ, and subtracts 1 whenever they are the same. A square is black if no negative number ever appears. Picture (b) has a black square wherever digits at more than half the possible positions differ between the *x* and *y* coordinates. Picture (c) has a black square wherever the maximum run of either identical or different digits has a length which is an odd number. All the patterns shown have the kind of intricate substructure typical of nesting. But none of the patterns are purely nested.



▪ **Page 610 · Generating functions.** A convenient algebraic way to describe a sequence of numbers *a[n]* is to give a generating function *Sum[a[n] x^n, {n, 0, ∞}]*. *1/(1 - x)* thus corresponds to the constant sequence and *1/(1 - x - x^2)* to the Fibonacci sequence (see page 890). A 2D array can be described by *Sum[a[t, n] x^n y^t, {n, -∞, ∞}, {t, -∞, ∞}]*. The array for rule 60 is then *1/(1 - (1 + x) y)*, for rule 90 *1/(1 - (1/x + x) y)*, for rule 150 *1/(1 - (1/x + 1 + x) y)* and for second-order reversible rule 150 (see page 439) *1/(1 - (1/x + 1 + x) y - y^2)*. Any rational function is the generating function for some additive cellular automaton.

▪ **Page 611 · Pascal's triangle.** See notes on page 870.

▪ **Nesting in bitwise functions.** See page 871.

▪ **Trinomial coefficients.** The coefficient of x^n in the expansion of $(1 + x + x^2)^t$ is

```
Sum[Binomial[n + t - 1 - 3 k, n - 3 k]
    Binomial[t, k] (-1)^k, {k, 0, t}]
```

which can be evaluated as

```
Binomial[2 t, n] Hypergeometric2F1[-n, n - 2 t, 1/2 - t, 1/4]
```

or finally *GegenbauerC[n, -t, -1/2]*. This result follows directly from the generating function formula

```
(1 - 2 x z + x^2)^-m == Sum[GegenbauerC[n, m, z] x^n, {n, 0, ∞}]
```

▪ **Gegenbauer functions.** Introduced by Leopold Gegenbauer in 1893 *GegenbauerC[n, m, z]* is a polynomial in *z* with integer coefficients for all integer *n* and *m*. It is a special case of *Hypergeometric2F1* and *JacobiP* and satisfies a second-order ordinary differential equation in *z*. The *GegenbauerC[n, d/2 - 1, z]* form a set of orthogonal functions on a *d*-dimensional sphere. The *GegenbauerC[n, 1/2, z]* obtained for *d = 3* are *LegendreP[n, z]*.

▪ **Standard mathematical functions.** There are an infinite number of possible functions with integer or continuous

arguments. But in practice there is a definite set of standard named mathematical functions that are considered reasonable to include as primitives in formulas, and that are implemented as built-in functions in *Mathematica*. The so-called elementary functions (logarithms, exponentials, trigonometric and hyperbolic functions, and their inverses) were mostly introduced before about 1700. In the 1700s and 1800s another several hundred so-called special functions were introduced. Most arose first as solutions to specific differential equations, typically in physics and astronomy; some arose as products, sums of series or inverses of other functions. In the mid-1800s it became clear that despite their different origins most of these functions could be viewed as special cases of `Hypergeometric2F1[a, b, c, z]`, and that the functions covered the solutions to all linear differential equations of a certain type. (`Zeta` and `PolyLog` are parametric derivatives of `Hypergeometric2F1`; elliptic modular functions are inverses.) Rather few new special functions have been introduced over the past century. The main reason has been that the obvious generalizations seem to yield classes of functions whose properties cannot be worked out with much completeness. So, for example, if there are more parameters it becomes difficult to find continuous definitions that work for all complex values of these parameters. (Typically one needs to generalize formulas that are initially set up with integer numbers of terms; examples include taking `Power[x, y]` to be `Exp[Log[x] y]` and `x!` to be `Gamma[x + 1]`.) And if one modifies the usual hypergeometric equation `y''[x] == f[y[x], y'[x]]` by making `f` nonlinear then solutions typically become hard to find, and vary greatly in character with the form of `f`. (For rational `f` Paul Painlevé in the 1890s identified just 6 additional types of functions that are needed, but even now series expansions are not known for all of them.) Generalizations of special functions can in principle be used to represent the results of many kinds of computations. Thus, for example, generalized elliptic theta functions represent solutions to arbitrary polynomial equations, while multivariate hypergeometric functions represent arbitrary conformal mappings. In *Mathematica*, however, functions like `Root` provide more convenient ways to access such results.

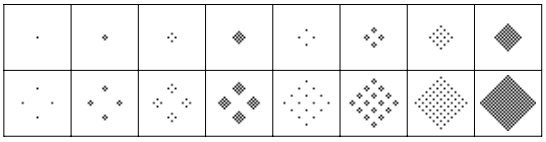
A variety of standard mathematical functions with integer arguments were introduced in the late 1800s and early 1900s in connection with number theory. A few functions that involve manipulation of digits have also become standard since the use of computers became widespread.

▪ **1D sequences.** Generating functions that are rational always lead to sequences which after reduction modulo 2 are purely repetitive. Algebraic generating functions can also lead to nested sequences. (Note that to get only integer sequences such generating functions have to be specially chosen.) `Sqrt[1 - 4 x]/2` yields a sequence with 1's at positions 2^m, as essentially obtained from the substitution system `{2 → {2, 1}, 1 → {1, 0}, 0 → {0, 0}}`. `Sqrt[(1 - 3 x)/(1 + x)]/2` yields sequence (a) on page 84. `(1 + Sqrt[(1 - 3 x)/(1 + x)])/(2 (1 + x))` (see page 890) yields the Thue-Morse sequence. (This particular generating function satisfies the equation `(1 + x)^3 f^2 - (1 + x)^2 f + x == 0`.) `(1 - 9 x)^1/3` yields almost the Cantor set sequence from page 83. `EllipticTheta[3, π, x]/2` gives a sequence with 1's at positions m^2.

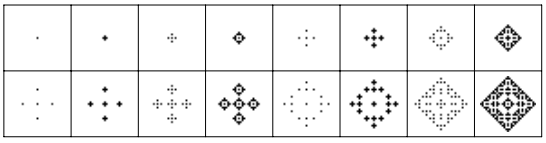
For any sequence with an algebraic generating function and thus for any nested sequence the n^{th} element can always be expressed in terms of hypergeometric functions. For the Thue-Morse sequence the result is

```
1/2 (-1)^n + (-3)^n √π  Hypergeometric2F1[3/2,
    -n, 3/2 - n, -1/3]/(4 n! Gamma[3/2 - n])
```

▪ **Multidimensional additive rules.** The 2D analog of rule 90 yields the patterns shown below. The colors of cells are given essentially by `Mod[Multinomial[t, x, y], 2]`. In d dimensions `(2 d)^DigitCount[t, 2, 1]` cells are black at step t. The fractal dimension of the `(d+1)`-dimensional structure formed from all black cells is `Log[2, 1 + 2 d]`.

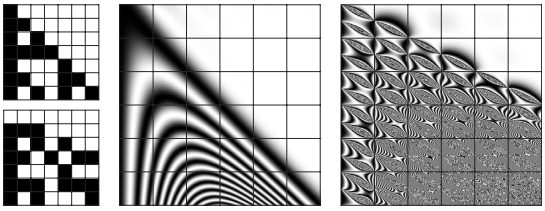


The 2D analog of rule 150 yields the patterns below; the fractal dimension of the structure in this case is `Log[2, (1 + Sqrt[1 + 4/d]) d]`.

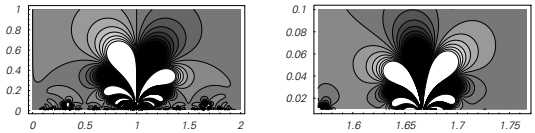


▪ **Continuous generalizations.** Functions such as `Binomial[t, n]` and `GegenbauerC[n, -t, -1/2]` can immediately be evaluated for continuous t and n. The pictures on the right below show `Sin[1/2 π a[t, n]]^2` for these functions (equivalent to `Mod[a[t, n], 2]` for integer `a[t, n]`). The discrete results on the left can be obtained by sampling only where integer grid lines cross. Note that without further conditions the continuous forms cannot be considered unique extensions of the discrete ones. The presence of poles in quantities such as

GegenbauerC[1/2, -t, -1/2] leads to essential singularities in the rightmost picture below. (Compare page 922.)



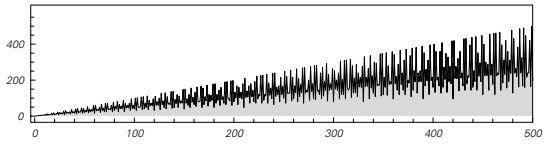
■ **Nested continuous functions.** Most standard continuous mathematical functions never show any kind of nested behavior. Elliptic theta and elliptic modular functions are exceptions. Each of these functions has definite finite values only in a limited region of the complex plane, and on the boundary of this region they exhibit singularities at every single rational point. The picture below shows *Im[ModularLambda[x + i y]]*. Like other elliptic modular functions, *ModularLambda* satisfies *f[z] == f[(a + b z)/(c + d z)]* with *a*, *b*, *c*, *d* integers such that *a c - b d == 1*. The function can be obtained as the solution to a second-order nonlinear ordinary differential equation. Nested behavior is also found for example in *EllipticTheta[3, 0, z]*, which is given essentially by $Sum[z^{n^2}, \{n, \infty\}]$.



■ **Page 613 · GCD array.** (See also page 950.) There are various deviations from perfect randomness. The density of white squares is asymptotically $6/\pi^2 \simeq 0.61$. (The probability for *s* randomly chosen integers to be relatively prime is *1/Zeta[s]*.) No 2×2 or larger block of white squares can ever occur. An arrangement of black squares with any list of relative offsets will always eventually occur. (This follows from the Chinese Remainder Theorem.) The first 2×2 block of black squares occurs at *{14, 20}*, the first 3×3 block at *{1274, 1308}* and the first 4×4 block at *{7247643, 10199370}*. The densities of such blocks are respectively about 0.002, 2×10^{-6} and 10^{-14}. In general the density for an arrangement of white squares with offsets *v* is given in *s* dimensions by (no simple closed formula seems to exist except for the 1×1 case)

```
Product[With[{p = Prime[n]},
    1 - Length[Union[Mod[v, p]]]/p^s], {n, ∞}]
```

White squares correspond to lattice points that are directly visible from the origin at the top left of the picture, so that lines to them do not pass through any other integer points. On row *n* the number of white squares encountered before reaching the leading diagonal is *EulerPhi[n]*. This function is shown below. Its computation is known in general to be equivalent in difficulty to factoring *n* (see page 1090). *GCD* can be computed using Euclid's algorithm as discussed on page 915.



■ **Power cellular automata.** Multiplication by *m* in base *k* corresponds to a local cellular automaton operation on digit sequences when every prime that divides *m* also divides *k*. The first non-trivial cases for which this is so are $k = 6$, $m = 2^i 3^j$ and $k = 10$, $m = 2^i 5^j$. When *m* itself divides *k*, the cellular automaton rule is *{_, b_, c_} → m Mod[b, k/m] + Quotient[c, k/m]*; in other cases the rule can be obtained by composition. A similar result holds for rational *m*, obtained for example by allowing *i* and *j* above to be negative. In all cases the cellular automaton rule, like the original operation on numbers, is invertible. The inverse rule, corresponding to multiplication by *1/m*, can be obtained by applying the rule for multiplication by the integer k^q/m, then shifting right by *q* positions. (See page 903.)

The condition for locality in negative bases (see page 902) is more stringent. The first non-trivial example is $k = -6$, $m = 8$, corresponding to a rule that depends on four neighboring cells.

Non-trivial examples of multiplication by *m* in base *k* all appear to be class 3 systems (see page 250), with small changes in initial conditions growing at a roughly fixed rate.

■ **Page 615 · Computing powers.** The method of repeated squaring (also known as the binary power method, Russian peasant method and Pingala's method) computes the quantity m^t by performing about *Log[t]* multiplications and building up the sequence

```
FoldList[#1^2 m^#2 &, 1, IntegerDigits[t, 2]]
```

(related to the Horner form for the base 2 representation of *t*). Given two numbers *x* and *y* their product can be computed in base *k* by (*FromDigits* does the carries)

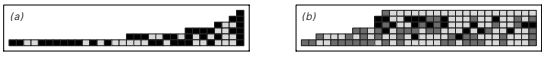
```
FromDigits[ListConvolve[IntegerDigits[x, k],
    IntegerDigits[y, k], {1, -1}, 0], k]
```

For numbers with *n* digits direct evaluation of the convolution would take about n^2 steps. But FFT-related methods reduce this to about *n Log[n]* steps (see also page 1142). And this implies that to find a particular digit of m^t in base *k* will take altogether about $t\,Log[t]^2$ steps.

One might think that a more efficient approach would be to start with the trivial length t digit sequence for c^t in base c, then to find a particular base k digit just by converting to base k. However, the straightforward method for converting a t-digit number x to base k takes about t divisions, though this can be reduced to around $Log[t]$ by using a recursive method such as

```
FixedPoint[Flatten[Map[If[# < k, #, With[
    {e = Ceiling[Log[k, #]/2]}, {Quotient[#, k^e], With[
      {s = Mod[#, k^e]}, If[s == 0, Table[0, {e}], {Table[0,
        {e - Floor[Log[k, s]] - 1}], s}]]]] &, #]] &, {x}]
```

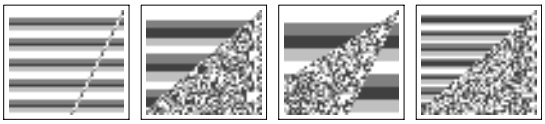
The pictures below show stages in the computation of 3^{20} (a) by a power tree in base 2 and (b) by conversion from base 3. Both approaches seem to require about the same number of underlying steps. Note that even though one may only want to find a single digit in m^t, I know of no way to do this without essentially computing all the other digits in m^t as well.



■ **Complex powers.** The pictures below show successive powers of complex numbers z with digits extracted according to

```
(2 d[Re[#], w] + d[Im[#], w] &)[z^t]

d[x_, w_] := If[x < 0, 1 - d[-x, w], IntegerDigits[x, 2, w]]
```

Non-trivial cases of complex number multiplication never correspond to local cellular automaton operations. (Compare page 933.)



■ **Additive cellular automata.** As discussed on page 951 a step in the evolution of an additive cellular automaton can be thought of as multiplication by a polynomial modulo k. After t steps, therefore, the configuration of such a system is given by $PolynomialMod[poly^t, k]$. This quantity can be computed using power tree methods (see below), though as discussed on page 609, even more efficient methods are also available. (A similar formalism can be set up for any of the cellular automata with generalized additivity discussed on page 952; see also page 886.)

■ **The more general case.** One can think of a single step in the evolution of any system as taking a rule r and state s, and producing a new state $h[r, s]$. Usually the representations that are used for r and s will be quite different, and the function h will have no special properties. But for both multiplication rules and additive cellular automata it turns out that rules and states can be represented in the same way, and the evolution functions h have the property of being associative, so that $h[a, h[b, c]] == h[h[a, b], c]$. This means that in effect one can always choose to evolve the rule rather than a state. A consequence is that for example 4 steps of evolution can be computed not only as $h[r, h[r, h[r, h[r, s]]]]$ but also as $h[h[h[r, r], h[r, r]], s]$ or $u = h[r, r]; h[h[u, u], s]$—which requires only 3 applications of h. And in general if h is associative the result $Nest[h[r, #] &, s, t]$ of t steps of evolution can be rewritten for example using the repeated squaring method as

```
h[Fold[If[#2 == 0, h[#1, #1], h[r, h[#1, #1]]] &,
    r, Rest[IntegerDigits[t, 2]]], s]
```

which requires only about $Log[t]$ rather than t applications of h.

As a very simple example, consider a system which starts with the integer 1, then at each step just adds 1. One can compute the result of 9 steps of evolution as $1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$, but a better scheme is to use partial results and compute successively $1 + 1; 2 + 2; 1 + 4; 5 + 5$—which is what the repeated squaring method above does when $h = Plus$, $r = s = 1$. This same basic scheme can be used with any associative function h—Max, GCD, And, Dot, $Join$ or whatever—so long as suitable forms for r and s are used.

For the multiplication rules discussed in the main text both states and rules can immediately be represented by integers, with $h = Times$, and $r = m$ giving the multiplier. For additive cellular automata, states and rules can be represented as polynomials (see page 951), with $h[a_, b_] := PolynomialMod[a\,b, k]$ and for example $r = 1 + x$ for elementary rule 60. The correspondence between multiplication rules and additive cellular automata can be seen even more directly if one represents all states by integers and computes h in terms of base k digits. In both cases it then turns out that h can be obtained from (see note above)

```
h[a_, b_] := FromDigits[g[ListConvolve[
      IntegerDigits[a, k], IntegerDigits[b, k], {1, -1}, 0]], k]
```

where for multiplication rules $g = Identity$ and for additive cellular automata $g = Mod[#, k] &$. For multiplication rules, there are normally carries (handled by $FromDigits$), but for power cellular automata, these have only limited range, so that $g = Mod[#, k^\sigma] &$ can be used.

For any associative function h the repeated squaring method allows the result of t steps of evolution to be computed with only about $Log[t]$ applications of h. But to be able to do this some of the arguments given to h inevitably need to be larger.

So whether a speedup is in the end achieved will depend on how fast *h* can be evaluated for arguments of various sizes. Typically the issue is whether *h[a, b]* for large *a* and *b* can be found with much less effort than it would take to evaluate *h[r, b]* about *a* times. If *h = Times*, then as discussed in the note above, the most obvious procedure for evaluating *h[a, b]* would involve about *m n* operations, where *m* and *n* are the numbers of digits in *a* and *b*. But when $m \simeq n$ FFT-related methods allow this to be reduced to about *n Log[n]* operations. And in fact whenever *h* is commutative (*Orderless*) it turns out that such methods can be used, and substantial speedups obtained. But whether anything like this will work in other cases is not clear.

(See also page 886.)

▪ **Evaluation chains.** The idea of building up computations like *1 + 1 + 1 + …* from partial results has existed since Egyptian times. Since the late 1800s there have been efforts to find schemes that require the absolute minimum number of steps. The method based on *IntegerDigits* in the previous two notes can be improved (notably by power tree methods), but apparently about *Log[t]* steps are always needed. (Finding the optimal addition chain for given *t* may be NP-complete.)

One can also consider building up lists of non-identical elements, say by successively using *Join*. In general a length *n* list can require about *n* steps. But if the list contains a nested sequence, say generated using a substitution system, then about *Log[n]* steps should be sufficient. (Compare page 566.)

▪ **Boolean formulas.** A Boolean function of *n* variables can always be specified by an explicit table giving values for all 2^n possible inputs. (Any cellular automaton rule with an *n*-cell neighborhood corresponds to such a function; digit sequences in rule numbers correspond to explicit tables of values.) Like ordinary algebraic functions, Boolean functions can also be represented by a variety of kinds of formulas. Those on pages 616 and 618 use so-called disjunctive normal form (DNF) *And[…] ∨ And[…] ∨ …*, which is common in practice in programmable logic arrays (PLAs). (The addition and multiplication operators in the main text should be interpreted as *Or* and *And* respectively.) In general any given function will allow many DNF representations; minimal ones can be found as described below. Writing a Boolean function in DNF is the rough analog of applying *Expand* to a polynomial. Conjunctive normal form (CNF) *Or[…] ∧ Or[…] ∧ …* is the rough analog of applying *Factor*. DNF and CNF both involve Boolean formulas of depth 2. As in the note on multilevel formulas below, one can also in effect introduce intermediate variables to get recursive formulas of larger depth, somewhat analogous to results from *Collect*. (Unbalanced depths in different parts of a formula lead to latencies in a circuit, reducing practical utility.)

▪ **DNF minimization.** From a table of values for a Boolean function one can immediately get a DNF representation just by listing cases where the value is 1. For one step in rule 30, for example, this yields *{{1, 0, 0}, {0, 1, 1}, {0, 1, 0}, {0, 0, 1}}*, as shown on page 616. One can think of this as specifying corners that should be colored on an *n*-dimensional Boolean hypercube. To reduce the representation, one must introduce "don't care" elements _; in this example the final minimal form consists of the list of 3 so-called implicants *{{1, 0, 0}, {0, 1, _}, {0, _, 1}}*. In general, an implicant with *m* _'s can be thought of as corresponding to an *m*-dimensional hyperplane on the Boolean hypercube. The problem of minimization is then to find the minimal set of hyperplanes that will cover the corners for a particular Boolean function. The first step is to work out so-called prime implicants corresponding to hyperplanes that cannot be contained in higher-dimensional ones. Given an original DNF list *s*, this can be done using *PI[s, n]*:

```
PI[s_, n_] := Union[Flatten[
      FixedPointList[f[Last[#], n] &, {{}, s}][[All, 1]], 1]]
g[a_, b_] := With[{i = Position[Transpose[{a, b}], {0, 1}]},
    If[Length[i] == 1 && Delete[a, i] === Delete[b, i],
      {ReplacePart[a, _, i]}, {}]]
f[s_, n_] := With[
  {w = Flatten[Apply[Outer[g, #1, #2, 1] &, Partition[Table[
        Select[s, Count[#, 1] == i &], {i, 0, n}], 2, 1], {1}],
    3]}, {Complement[s, w, SameTest → MatchQ], w}]
```

The minimal DNF then consists of a collection of these prime implicants. Sometimes it is all of them, but increasingly often when $n \geq 3$ it is only some. (For example, in *{{0, 0, _}, {0, _, 1}, {_, 0, 0}}* the first prime implicant is covered by the others, and can therefore be dropped.) Given the original list *s* and the complete prime implicant list *p* the so-called Quine-McCluskey procedure can be used to find a minimal list of prime implicants, and thus a minimal DNF:

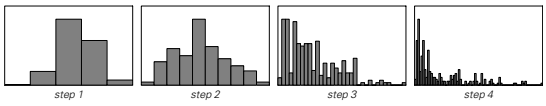
```
QM[s_, p_] := First[Sort[Map[p[[#]] &,
      h[{}, Range[Length[s]], Outer[MatchQ, s, p, 1]]]]]
h[i_, r_, t_] := Flatten[Map[h[Join[i, r[[#]]], Drop[r, #],
      Delete[Drop[t, {}, #], Position[t[[All, #]], {True}]]] &,
    First[Sort[Map[Position[#, True] &, t]]]], 1]
h[i_, _, {}] := {i}
```

The number of steps required in this procedure can increase exponentially with the length of *p*. Other procedures work slightly more efficiently, but in general the problem of finding the minimal DNF for a Boolean function of *n*

variables is NP-complete (see page 768) and is thus expected to grow in difficulty faster than any polynomial in n. In practice, however, cases up to about $n = 12$ are nevertheless currently handled quite routinely.

■ **Formula sizes.** There are a total of 2^{2^n} possible Boolean functions of n variables. The maximum number of terms needed to represent any of these functions in DNF is 2^{n-1}. The actual numbers of functions which require 0, 1, 2, … terms is for $n = 2$: *{1, 9, 6}*; for $n = 3$: *{1, 27, 130, 88, 10}*, and for $n = 4$: *{1, 81, 1804, 13472, 28904, 17032, 3704, 512, 26}*. The maximal length turns out always to be realized for the simple parity function *Xor*, as well as its negation. The reason for this is essentially that these functions are the ones that make the coloring of the Boolean hypercube maximally fragmented. (Other functions with maximal length are never additive, at least for $n \le 4$.)

■ **Cellular automaton formulas.** See page 869. The maximum length DNF for elementary rules after 1 step is 4, and this is achieved by rules 105, 107, 109, 121, 150, 151, 158, 182, 214 and 233. These rules have behavior of quite varying complexity. Rules 150 and 105 are additive, and correspond to *Xor* and its negation. After t steps the maximum conceivable DNF would be of length 2^{2t}. In practice, after 2 steps, the maximum length is 9, achieved by rules 107, 121 and 182; after 3 steps, it is 33 achieved by rule 182; after 4 steps, 78 achieved by rule 129; after 5 steps 256 achieved by rules 105 and 150. The distributions of lengths for all elementary rules are shown below.



Note that the length of a minimal DNF representation cannot be considered a reliable measure of the complexity of a function, since among other things, just exchanging the role of black and white can substantially change this length (as in the case of rule 126 versus rule 129).

■ **Primitive functions.** There are several possible choices of primitive functions that can be combined to represent any Boolean function. In DNF *And*, *Or* and *Not* are used. *Nand = Not[And[##]] &* alone is also sufficient, as shown on page 619 and further discussed on page 807. (It is indicated by $\bar{\pi}$ in the main text.) The functions *And*, *Xor* and *Not* are equivalent to *Times*, *Plus* and *1 - # &* for variables modulo 2, and in this case algebraic functions like *PolynomialReduce* can be used for minimization. (See also page 1102.)

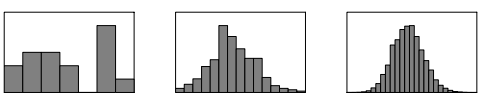
■ **Multilevel formulas.** DNF formulas always have depth 2. By allowing larger depths one can potentially find smaller formulas for functions. A major result from the 1980s is that it requires a formula with depth at least *Log[n]/(c + Log[Log[n]])* to make it possible to represent an *Xor* of n variables using a polynomial number of *And*, *Or* and *Not* operations. If one chooses an n-variable Boolean function at random out of the 2^{2^n} possibilities, it is typical that regardless of depth a formula involving at least $2^n/n$ operations will be needed to represent it. A formula of polynomial size and logarithmic depth exists only when a function is the computational complexity class NC discussed on page 1149.

Little is known about systematic minimization of Boolean formulas with depths above 2. Nevertheless, some programs for circuit design such as SIS do include a few heuristics. And this for example allows SIS to generate higher depth formulas somewhat smaller than the minimal DNF for the first three steps of rule 30 evolution.

$b_1 = a_2 + a_3;\ \bar{a}_1 b_1 + a_1 \bar{b}_1$
$b_1 = \bar{a}_2 a_3 + a_2 \bar{a}_3;\ b_2 = a_4 + a_5;\ a_1 b_1 + a_1 \bar{b}_2 + \bar{a}_1 \bar{b}_1 b_2$
$b_1 = a_6 + a_7;\ b_2 = a_4 + a_5;\ b_3 = \bar{a}_6 b_1 + a_4 \bar{b}_1;\ b_4 = \bar{b}_1 + b_2;\ \bar{a}_1 \bar{a}_3 b_3 + \bar{a}_1 a_2 \bar{b}_2 + a_1 \bar{a}_2 \bar{a}_3 \bar{b}_3 + \bar{a}_1 a_2 a_4 \bar{b}_3 + a_1 a_2 \bar{a}_4 b_2 + \bar{a}_1 \bar{a}_2 a_3 b_4 + a_1 \bar{a}_2 a_3 \bar{b}_4 + a_1 a_2 a_3 b_3 b_4$

■ **Page 619 · NAND expressions.** If one allows a depth of at most $2n$ any n-input Boolean function can be obtained just by combining 2-input *Nand* functions. (See page 807.) (Note that unless one introduces an explicit copy operation—or adds variables as in the previous note—there is no way to use the same intermediate result multiple times without recomputing it.)

The pictures below show the distributions of numbers of *Nand* operations needed for all 2^{2^n} n-input Boolean functions. For $n = 2$, the largest number of such operations is 6, achieved by *Nor*; for $n = 3$, it is 14, achieved by *Xor* (rule 150); for $n = 4$, it is 27, achieved by rule 5737, which is *Not[Xor[##]] &* except when all inputs are *True*. The average number of operations needed when $n = 2, 3, 4$ is about *{2.875, 6.09, 12.23}*.

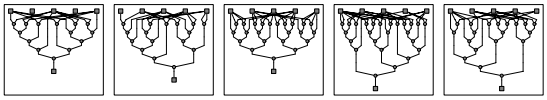


The maximum depths for the expressions of minimal size are respectively 4, 6 and 7, always achieved among others for the function taking the most *Nand* operations. The total numbers of functions involving successive depths are: $n = 2$: *{2, 3, 5, 6}*, $n = 3$: *{3, 6, 22, 99, 72, 54}*, $n = 4$: *{4, 10, 64, 923, 9663, 54622, 250}*, corresponding to averages *{2.9, 4.5, 5.8}*.

The following generates explicit lists of n-input Boolean functions requiring successively larger numbers of *Nand* operations:

```
Map[FromDigits[#, 2] &, NestWhile[Append[#,
    Complement[Flatten[Table[Outer[1 - Times[##] &,
      #[[i]], #[[-i]], 1], {i, Length[#]}], 2], Flatten[#, 1]]] &,
  {1 - Transpose[IntegerDigits[Range[2^n] - 1, 2, n]]},
  Length[Flatten[#, 1]] < 2^2^n &], {2}]
```

The results for 2-step cellular automaton evolution in the main text were found by a recursive procedure. First, expressions containing progressively more *Nand* operations were enumerated, and those for functions that had not been seen before were kept. It then turned out that this made it possible to get to expressions at least half as large as any needed, so that it could be assumed that remaining expressions could be decomposed as `f[##] ⊼ g[##] &`, where f had already been found. The pictures below show some more results obtained in this way.



rule 150 rule 110 rule 126 rule 45 rule 54

■ **Cellular automaton formulas.** For 1 step, the elementary cellular automaton rules are exactly the 256 $n = 3$ Boolean functions. For 2 steps, they represent a small subset of the 2^{32} $n = 5$ functions. They require an average of about 11.6 *Nand* operations, and a maximum of 27 (achieved by rules 107 and 121).

For rule 254 the result after t steps (which is always asymmetric, even though the rule is symmetric) is

```
Nest[{{#, #[[2]] + 1}, #[[2]] + 1} &, {{1, 1}, {2, 2}}, t - 2]
```

If explicit copy operations were allowed, then the number of *Nand* operations after t steps could not increase faster than t^2 for any rule. But without copy (fanout) operations no corresponding result is immediately clear.

■ **Binary decision diagrams.** One can specify a Boolean function of n variables by giving a finite automaton (and thus a network) in which paths exist only for those lists of values for which the function yields *True*. The resulting so-called binary decision diagram (BDD) can be minimized using the methods of page 957. Out of all possible Boolean functions the number that require BDDs of sizes 1, 2, … is for $n = 2$: `{1, 0, 6, 9}` and for $n = 3$: `{1, 0, 0, 27, 36, 132, 60}`; the absolute maximum grows roughly like 2^n. For cellular automata with simple behavior, the minimal BDD typically grows linearly on successive steps. For rule 254, for example, it is $8\,t + 2$, while for rule 90 it is $4\,t + 2$. For cellular automata with more complex behavior, it typically grows roughly exponentially. Thus for rule 30 it is `{7, 14, 29, 60, 129}` and for rule 110 `{7, 15, 27, 52, 88}`. The size of the minimal BDD can depend on the order in which variables are specified; thus for example, just reflecting rule 30 to give rule 86 yields `{6, 11, 20, 36, 63}`.

In practical system design BDDs have become fairly popular in the past ten years, and by maintaining minimality when logical combinations of functions are formed, cases with millions of nodes have been studied. (Some practical systems are found to yield fairly small BDDs, while others are not.)

■ **History.** Logic has been used as an abstraction of arguments in ordinary language since antiquity. Its serious mathematical formulation began with the work of George Boole in the mid-1800s. (See page 1151.) Concepts of Boolean algebra were applied to electronic switching circuits by Claude Shannon in 1937, and became a standard part of electronic design methodology by the 1950s. DNF had been introduced as part of the development of mathematical logic in the early 1900s, but became particularly popular in the 1970s with the advent of programmable logic arrays (PLAs) used in application-specific integrated circuits (ASICs). Diagrammatic and mechanical methods for minimizing simple logic expressions have existed since at least medieval times. More systematic methods for minimizing complex expressions began to be developed in the early 1950s, but until well into the 1980s a diagrammatic method known as a Karnaugh map was the most commonly used in practice. In the late 1970s there began to be computer programs for large-scale Boolean minimization—the best known being *Espresso*. Only in the 1990s, however, did exact minimization of complex DNF expressions become common. Minimization of Boolean expressions with depth larger than 2 has been considered off and on since the late 1950s, and became popular in the 1990s in connection with the BDDs discussed above. Various forms of Boolean minimization have routinely been used in chip and circuit design since the late 1980s, though often physical and geometrical constraints are now more important than pure logical ones. In addition, theoretical studies of minimal Boolean circuits became increasingly popular starting in the 1980s, as discussed on page 1148.

■ **Reversible logic.** In an ordinary Boolean function with n inputs there is no unique way to tell from its output which of the 2^n possible sets of inputs was given. But as noted in the 1970s, it is possible to set up systems that evaluate Boolean functions, yet operate reversibly. The basic idea is to have m outputs as well as m inputs—with every one of the 2^m possible sets of inputs mapping to a unique set of outputs. Normally one specifies the first n inputs, taking the others to be fixed, and then looks say at the first output, ignoring all others. One can represent the inside of such a system much

like a sorting network from page 1142—but with s-input s-output gates instead of pair comparisons. If each such gate is itself reversible, then overall reversibility is guaranteed. With gates that in effect implement $\{p, q\} \to \{p \bar{\pi} q\}$ and $\{p\} \to \{p, p\}$ (with other inputs constant, and other outputs ignored) one can set up a direct translation of Boolean functions given in the form shown on page 619. Of the 24 possible reversible $s = 2$ gates, none can yield anything other than additive Boolean functions (as formed from *Xor* and *Not*). But of the 40,320 ($8!$) reversible $s = 3$ gates (in 52 distinct classes) it turns out that 38,976 (in 23 classes) can be used to reproduce any possible Boolean function. A simple example of such a universal gate is $\{p\_, q\_, 1\} \to \{q, p, 1\}$—and not allowing permutations of gate inputs (or in effect wire crossings) a simple example is $\{p\_, q\_, q\_\} \to \{q, 1 - p, 1 - p\}$. (Compare pages 1147 and 1173.)

■ **Continuous systems.** The systems I discuss in the main text of this section are mostly discrete. But from experience with traditional mathematics one might have the impression that it would at some basic level be easier to get formulas for continuous systems. I believe, however, that this is not the case, and that the reason for the impression is just that it is usually so much more difficult even to represent the states of continuous systems that one normally tends to work only with ones that have comparatively simple overall behavior—and are therefore more readily described by formulas. (See also pages 167 and 729.)

As an example of what can happen in continuous systems consider iterated mappings $x \to a\, x\, (1 - x)$ from page 920. Each successive step in such a mapping can in principle be represented by an algebraic formula. But the table below gives for example the actual algebraic formulas obtained in the case $a = 4$ after applying *FullSimplify*—and shows that these increase quite rapidly in complexity.

x
$4\,(1 - x)\,x$
$16\,(1 - 2\,x)^2\,(1 - x)\,x$
$64\,(1 - 2\,x)^2\,(1 - x)\,x\,(8\,(x - 1)\,x + 1)^2$
$256\,(1 - 2\,x)^2\,(1 - x)\,x\,(8\,(x - 1)\,x + 1)^2\,(32\,(x - 1)\,x\,(1 - 2\,x)^2 + 1)^2$
$1024\,(1 - 2\,x)^2\,(1 - x)\,x\,(8\,(x - 1)\,x + 1)^2\,(32\,(x - 1)\,x\,(1 - 2\,x)^2 + 1)^2$ $(128\,(1 - 2\,x)^2\,(x - 1)\,x\,(8\,(x - 1)\,x + 1)^2 + 1)^2$

In the specific case $a = 4$, however, it turns out that by allowing more sophisticated mathematical functions one can get a complete formula: the result after any number of steps t can be written in any of the forms

$$\text{Sin}[2^t\, \text{ArcSin}[\sqrt{x}\,]]^2$$

$$(1 - \text{Cos}[2^t\, \text{ArcCos}[1 - 2\,x]])/2$$

$$(1 - \text{ChebyshevT}[2^t,\, 1 - 2\,x])/2$$

where these follow from functional relations such as

$$\text{Sin}[2\,x]^2 == 4\,\text{Sin}[x]^2\,(1 - \text{Sin}[x]^2)$$

$$\text{ChebyshevT}[m\,n,\, x] == \text{ChebyshevT}[m,\, \text{ChebyshevT}[n,\, x]]$$

For $a = 2$ it also turns out that there is a complete formula:

$$(1 - (1 - 2\,x)^{2^t})/2$$

And the same is true for $a = -2$:

$$1/2 - \text{Cos}[1/3\,(\pi - (-2)^t\,(\pi - 3\,\text{ArcCos}[1/2 - x]))]$$

In all these examples t enters essentially only in a^t. And if one assumes that this is a general feature then one can formally derive for any a the result

$$1/2\,(1 - g[a^t\, \text{InverseFunction}[g][1 - 2\,x]])$$

where g is a function that satisfies the functional equation

$$g[a\,x] == 1 + 1/2\,a\,(g[x]^2 - 1)$$

When $a = 4$, $g[x]$ is $\text{Cosh}[\text{Sqrt}[2\,x]]$. When $a = 2$ it is $\text{Exp}[x]$ and when $a = -2$ it is $2\,\text{Cos}[1/3\,(\pi - \sqrt{3}\,x)]$. But in general for arbitrary a there is no standard mathematical function that seems to satisfy the functional equation. (It has long been known that only elliptic functions such as *JacobiSN* satisfy polynomial addition formulas—but there is no immediate analog of this for replication formulas.) Given the functional equation one can find a power series for $g[x]$ for any a. The series has an accumulation of poles on the circle $\text{Abs}[a]^2 == 1$; the coefficient of x^m turns out to have denominator

```
2^(m - DigitCount[m, 2, 1]) Apply[Times,
    Table[Cyclotomic[s, a]^Floor[(m - 1)/s], {s, m - 1}]]
```

For other iterated maps general formulas also seem rare. But for example $x \to a\,x + b$ and $x \to 1/(a + b\,x)$ both give results just involving powers, while $x \to \text{Sqrt}[a\,x + b]$ sometimes yields trigonometric functions, as on page 915. In addition, from a known replication formula for an elliptic or other function one can often construct an iterated map whose behavior can be expressed in terms of that function. (See also page 919.)

Human Thinking

■ **The brain.** There are a total of about 100 billion neurons in a human brain (see page 1075), each with an average of a few thousand synapses connecting it to other cells. On a small scale the arrangement of neurons seems quite haphazard. But on a larger scale the brain seems to be organized into areas with very definite functions. This organization is sometimes revealed by explicitly following nerve fibers. More often it has been deduced by looking at what happens if parts of the brain are disabled or stimulated. In recent times it has also begun to be possible to image local electrical and metabolic activity while the brain is in normal operation. From all these methods it is known that each kind of sensory input is first

processed in its own specific area of the brain. Inputs from different senses are integrated in an area that effectively maintains a map of the body; a similar area initiates output to muscles. Certain higher mental functions are known to be localized in definite areas of the brain, though within these areas there is often variability between individuals. Areas are currently known for specific aspects of language, memory (see below) and various cognitive tasks. There is some evidence that thinking about seemingly rather similar things can lead to significantly different patterns of activity.

Most of the action of the brain seems to be associated with local electrical connections between neurons. Some collective electrical activity is however revealed by EEG. In addition, levels of chemicals such as hormones, drugs and neurotransmitters can have significant global effects on the brain.

▪ **History.** Ever since antiquity immense amounts have been written about human thinking. Until recent centuries most of it was in the tradition of philosophy, and indeed one of the major themes of philosophy throughout its history has been the elucidation of principles of human thinking. However, almost all the relevant ideas generated have remained forever controversial, and almost none have become concrete enough to be applied in science or technology. An exception is logic, which was introduced in earnest by Aristotle in the 4^{th} century BC as a way to model certain patterns of human reasoning. Logic developed somewhat in medieval times, and in the late 1600s Gottfried Leibniz tried to use it as the foundation for a universal language to capture all systematic thinking. Beginning with the work of George Boole in the mid-1800s most of logic began to become more closely integrated with mathematics and even less convincingly relevant as a model for general human thinking.

The notion of applying scientific methods to the study of human thinking developed largely with the rise of the field of psychology in the mid-1800s. Two somewhat different approaches were taken. The first concentrated on doing fairly controlled experiments on humans or animals and looking at responses to specific stimuli. The second concentrated on trying to formulate fairly general theories based on observations of overall human behavior, initially in adults and later especially in children. Both approaches achieved some success, but by the 1930s many of their positions had become quite extreme, and the identification of phenomena to contradict every simple conclusion reached led increasingly to the view that human thinking would allow no simple explanations.

The idea that it might be possible to construct machines or other inanimate objects that could emulate human thinking existed already in antiquity, and became increasingly popular starting in the 1600s. It began to appear widely in fiction in the 1800s, and has remained a standard fixture in portrayals of the future ever since.

In the early 1900s it became clear that the brain consists of neurons which operate electrically, and by the 1940s analogies between brains and electrical machines were widely discussed, particularly in the context of the cybernetics movement. In 1943 Warren McCulloch and Walter Pitts formulated a simple idealized model of networks of neurons and tried to analyze it using methods of mathematical logic. In 1949 Donald Hebb then argued that simple underlying neural mechanisms could explain observed psychological phenomena such as learning. Computer simulations of neural networks were done starting in the mid-1950s, but the networks were too small to have any chance to exhibit behavior that could reasonably be identified with thinking. (Ironically enough, as mentioned on page 879, the phenomenon central to this book of complex behavior with simple underlying rules was in effect seen in some of these experiments, but it was considered a distraction and ignored.) And in the 1960s, particularly after Frank Rosenblatt's introduction of perceptrons, neural networks were increasingly used only as systems for specific visual and other tasks (see page 1076).

The idea that computers could be made to exhibit human-like thinking was discussed by Alan Turing in 1950 using many of the same arguments that one would give today. Turing made the prediction that by 2000 a computer would exist that could pass the so-called Turing test and be able to imitate a human in a conversation. (René Descartes had discussed a similar test for machines in 1637, but concluded that it would never be passed.) When electronic computers were first becoming widespread in the 1950s they were often popularly referred to as "electronic brains". And when early efforts to make computers perform tasks such as playing games were fairly successful, the expectation developed that general human-like thinking was not far away. In the 1960s, with extensive support from the U.S. government, great effort was put into the field of artificial intelligence. Many programs were written to perform specific tasks. Sometimes the programs were set up to follow general models of the high-level processes of thinking. But by the 1970s it was becoming clear that in almost all cases where programs were successful (notable examples being chess, algebra and autonomous control), they typically worked by following definite algorithms not closely related to general human thinking.

Occasional work on neural networks had continued through the 1960s and 1970s, with a few definite results being obtained

using methods from physics. Then in the early 1980s, particularly following work by John Hopfield, computer simulations of neural networks became widespread. Early applications, particularly by Terrence Sejnowski and Geoffrey Hinton, demonstrated that simple neural networks could be made to learn tasks of at least some sophistication. But by the mid-1990s it was becoming clear that—probably in large part as a consequence of reliance on methods from traditional mathematics—typical neural network models were mostly being successful only in situations where what was needed was a fairly straightforward extension of standard continuous probabilistic models of data.

■ **The future.** To achieve human-like thinking with computers will no doubt require advances in both basic science and technology. I strongly suspect that a key element is to be able to store a collection of experiences comparable to those of a human. Indeed, to succeed even with specific tasks such as speech recognition or language translation seems to require human-like amounts of background knowledge. Present-day computers are beginning to have storage capacities that are probably comparable to those of the brain. From looking at the brain one might guess that parallel or other non-standard hardware might be required to achieve efficient human-like thinking. But I rather suspect that—much as in the analogy between birds and airplanes—it will in the end be possible to set up algorithms that achieve the same basic functions but work satisfactorily even on standard sequential-processing computers.

■ **Sleep.** A common feature of higher organisms is the existence of distinct behavioral states of sleep and wakefulness. There are various theories that sleep is somehow fundamental to the process of thinking. But my guess is that its most important function is quite mundane: just as muscles build up lactic acid waste products, so also I suspect synapses in the brain build up waste products, and these can only safely be cleared out when the brain is not in normal use.

■ **Page 621 · Pointer encoding.** The pointer encoding compression method discussed on page 571 implements a very simple form of memory based on literal repetitions, and already leads to fairly good compression of many kinds of data.

■ **Page 622 · Hashing.** Given data in the form of sequences of numbers between *0* and *k - 1*, a very simple hashing scheme is just to compute *FromDigits[Take[list, n], k]*. But for data corresponding, say, to English words this scheme yields a very nonuniform distribution of hash codes, since, for example, there are many words beginning with "ba", but none beginning with "bb". The slightly modified but still very simple scheme *Mod[FromDigits[list, k], m]*, where *m* is usually chosen to be a prime, is what is most often used in practice. For a fair fraction of values of *m*, the hash codes obtained from this scheme change whenever any element of *list* is changed. If $m = k^s - 1$ then it turns out that interchanging a pair of adjacent length *s* blocks in *list* never affects the result. Out of the many hundreds of times that I have used hashing in practice, I recall only a couple of cases where schemes like the one just described were not adequate, and in these cases the data always turned out to have quite dramatic regularities.

In typical applications hash codes give locations in computer memory, from which actual data is found either by following a chain of pointers, or by probing successive locations until an empty one is reached. In the internals of *Mathematica* the most common way that hashing is used is for recognizing data and finding unique stored versions of it. There are several subtleties associated with setting up hash codes that appropriately handle approximate real numbers and *Mathematica* patterns.

Hashing is a sufficiently simple idea that it has been invented independently many times since at least the 1950s. The main alternative to hashing is to store data with successive elements corresponding to successive levels in a tree. In the past decade, hashing has become widely used not only for searching but also for authentication. The basic idea in this case is to take a document and to compute from it a small hash code that changes when almost any change is made in the document, and for which it is a difficult problem of cryptanalysis to work out what changes in the document will lead to no change in the hash code. Schemes for such hash codes can fairly easily be constructed using rule 30 and other cellular automata.

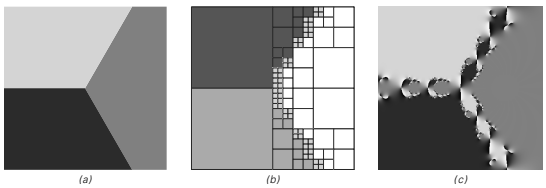
■ **Page 623 · Similar words.** The soundex system for hashing names according to sound was first used on 1880 U.S. census data, and is still today widely used by telephone information services. The system works essentially by dropping vowels and assigning consonants to six possible groups. More sophisticated systems along the same lines can be set up using finite automata.

Natural language query systems usually work by stripping words to their linguistic roots (e.g. "stripping" → "strip") before looking them up. Spell-checking systems typically find suggested corrections by doing a succession of lookups after applying transformations based on common errors.

Even given two specific words it can be difficult to find out whether they should be considered similar. Fairly efficient

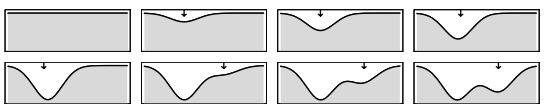
algorithms are known for cases such as genetic sequences where small numbers of insertions, deletions and substitutions are expected. But if more complicated transformations are allowed—say corresponding to rules in a multiway system—the problem rapidly becomes intractable (see page 765).

▪ **Numerical data.** In situations where pieces of data can be thought of as points in space similarity can often be defined in terms of spatial distance. And this means that around every point corresponding to a piece of data in memory there is a region of points that can be considered more similar to that point than to any other. Picture (a) shows a so-called Voronoi diagram (see page 1038) obtained in this way in two dimensions. Particularly in higher dimensions, it becomes rather difficult in practice to determine for certain which existing point is closest to some new point. But to do it approximately is considerably easier. One approach, illustrated in picture (b), is to use a d-dimensional tree. Another approach, illustrated in picture (c), is to set up a continuous function with minima at the existing points, and then to search for the closest minimum. In most cases, this search will be done using some iterative scheme such as Newton's method; the result is that the boundaries between regions typically take on an intricate nested form. (The case shown corresponds to iteration of the map $z \to z - (z^3 - 1)/(3 z^2)$ corresponding to Newton's method for finding the complex roots of $z^3 == 1$.)



(a) (b) (c)

The pictures below show how one can build up a kind of memory landscape by successively adding points. In a first approximation, the regions considered similar to a particular minimum are delimited by sharp watersheds corresponding to local maxima in the landscape. But if an iterative scheme for minimization is used, these watersheds are typically no longer sharp, but take on a local nested structure, much as in picture (c) above.



In numbers earlier digits are traditionally considered more important than later ones, and this allows numbers to be arranged in a simple one-dimensional sequence. But in strings where each element is considered equally important, no such layout is possible. A vague approximation, perhaps useful for some applications, is nevertheless to use a space-filling curve (see page 893).

▪ **Error-correcting codes.** In many information transmission and storage applications one needs to be able to recover data even if some errors are introduced into it. The standard way to do this is to set up an error-correcting code in which blocks of m original data elements are represented by a codeword of length n that in effect includes some redundant elements. Then—somewhat in analogy to retrieving closest memories—one can take a sequence of length n that one receives and find the codeword that differs from it in the fewest elements. If the codewords are chosen so that every pair differs by at least r elements (or equivalently, have so-called Hamming distance at least r), then this means that errors in up to *Floor[(r - 1)/2]* elements can be corrected, and finding suitable codewords is like finding packings of spheres in n-dimensional space. It is common in practice to use so-called linear codes which can be analyzed using algebraic methods, and for which the spheres are arranged in a repetitive array. The Hamming codes with $n = 2^s - 1$, $m = n - s$, $r = 3$ are an example, invented by Marcel Golay in 1949 and Richard Hamming in 1950. Defining

```
PM[s_] := IntegerDigits[Range[2^s - 1], 2, s]
```

blocks of data of length m can be encoded with

```
Join[data, Mod[data . Select[PM[s], Count[#, 1] > 1 &], 2]]
```

while blocks of length n (and at most one error) can be decoded with

```
Drop[(If[# == 0, data, MapAt[1 - # &, data, #]] &)[
    FromDigits[Mod[data . PM[s], 2], 2]], -s]
```

A number of families of linear codes are known, together with a few nonlinear ones. But in all cases they tend to be based on rather special mathematical structures which do not seem likely to occur in any system like the brain.

▪ **Matrix memories.** Many times since the 1950s it has been noted that methods from linear algebra suggest ways to construct associative memories in which data can potentially be retrieved on the basis of some form of similarity. Typically one starts from some list of vectors to be stored, then forms a matrix such as *m = PseudoInverse[list]*. Given a new piece of data corresponding to a vector v, its decomposition in terms of stored vectors can be found by computing $v \cdot m$. And by applying various forms of thresholding one can often pick out at least approximately the stored vector closest to the piece of data given. But such schemes tend to be inefficient in practice, as well as presumably being unrealistic as actual models of the brain.

■ **Neural network models.** The basic rule used in essentially all neural network models is extremely simple. Each neuron is assumed to have a value between -1 and 1 corresponding roughly to a firing rate. Then given a list *s[i]* of the values of one set of neurons, one finds the values of another set using *s[i + 1] = u[w . s[i]]*, where in early models *u = Sign* was usually chosen, and now *u = Tanh* is more common, and *w* is a rectangular matrix which gives weights—normally assumed to be continuous numbers, often between -1 and +1—for the synaptic connections between the neurons in each set. In the simplest case, studied especially in the context of perceptrons in the 1960s, one has only two sets of neurons: an input layer and an output layer. But with suitable weights one can reproduce many functions. For example, with three inputs and one output, *w = {{-1, +1, -1}}* yields essentially the rule for the rule 178 elementary cellular automaton. But out of the 2^{2^n} possible Boolean functions of *n* inputs, only 14 (out of 16) can be obtained for *n = 2*, 104 (out of 256) for *n = 3*, 1882 for *n = 4*, and 94304 for *n = 5*. (The VC dimension is *n + 1* for such systems.) The key idea that became popular in the early 1980s was to consider neural networks with an additional layer of "hidden units". By introducing enough hidden units it is then possible—just as in the formulas discussed on page 616—to reproduce essentially any function. Suitable weights (which are typically far from unique) are in practice usually found by gradient descent methods based either on minimization of deviations from desired outputs given particular inputs (supervised learning) or on maximization of some discrimination or other criterion (unsupervised learning).

Particularly in early investigations of neural networks, it was common to consider systems more like very simple cellular automata, in which the *s[i]* corresponded not to states of successive layers of neurons, but rather to states of the same set of neurons at successive times. For most choices of weights, such a system exhibits typical class 3 behavior and never settles down to give an obvious definite output. But in special circumstances probably not of great biological relevance it can yield class 2 behavior. An example studied by John Hopfield in 1981 is a symmetric matrix *w* with neuron values being updated sequentially in a random order rather than in parallel.

■ **Memory.** Since the early 1900s it has been suspected that long-term memory is somehow encoded in the strengths of synaptic connections between nerve cells. It is known that at least in specific cases such strengths can remain unchanged for at least hours or more, but can immediately change if connected nerve cells have various patterns of simultaneous excitation. The changes that occur appear to be associated changes in ionic channels in cell membranes and sometimes with the addition of new synapses between cells.

Observations suggest that in humans there are several different types of memory, with somewhat different characteristics. (Examples include memory for facts and for motor skills.) Usually there is a short-term or so-called working component, lasting perhaps 30 seconds, and typically holding perhaps seven items, and a long-term component that can apparently last a lifetime. Specific parts of the brain (such as the hippocampus) appear necessary for the long-term component to form. In at least some cases there is evidence for specialized areas that handle particular types of memories. When new data is first presented, many parts of the brain are often active in processing it. But once the data has somehow been learned, only parts directly associated with handling it usually appear to be active.

Memories often seem at some level to be built up incrementally, as reflected in smooth learning curves for motor skills. It is not clear whether this is due to actual incremental changes in nerve cells or just to the filling in of progressively more cases that differ in detail.

Experiments on human learning suggest that a particular memory typically involves an association between components from several sensory systems, as well as emotional state.

When several incomplete examples of data are presented, there appears to be some commonality in the character of generalizations that we make. One mathematically convenient but probably unrealistic model studied in recent years in the context of computational learning theory involves building up minimal Boolean formulas consistent with the examples seen.

■ **Child development.** As children get older their thinking becomes progressively more sophisticated, advancing through a series of fairly definite stages that appear to be associated with an increasing ability to handle generalization and abstraction. It is not clear whether this development is primarily associated with physiological changes or with the accumulation of more experiences (or, in effect, with the addition of more layers of software). Nor is it clear how it relates to the fact that the number of items that can be stored in short-term memory seems steadily to increase.

■ **Computer interfaces.** The earliest computer interfaces were essentially just numerical. By the 1960s text-based interfaces were common, and in the decade following the introduction of the Macintosh in 1984 graphical interfaces based on menus and dialogs came to largely dominate consumer software. Such interfaces work well if what one wants is basically to take a

single object and apply operations to it. And they can be extended somewhat by using visual block diagrams or flowcharts. But whenever there is neither just a single active data element nor an obvious sequence of independent execution steps—as for many of the programs in this book—my experience has always been that the only viable choice of interface is a computer language like *Mathematica*, based essentially on one-dimensional sequences of word-like constructs. The rule diagrams in this book represent a possible new method for specifying some simpler programs, but it remains to be seen whether such diagrams can readily both be created incrementally by humans and interpreted by computer.

▪ **Page 627 · Structure of *Mathematica*.** Beneath all the sophisticated capabilities of *Mathematica* lies a remarkably simple basic structure. The key idea is to represent data of any kind by a symbolic expression of the general form *head[arg$_1$, arg$_2$, …]*. ($a + b^2$ is thus *Plus[a, Power[b, 2]]*, *{a, b, c}* is *List[a, b, c]* and *a = b + 1* is *Set[a, Plus[b, 1]]*.) The basic action of *Mathematica* is then to transform such expressions according to whatever rules it knows. Most often these rules are specified in terms of *Mathematica* patterns—expressions in which _ can stand for any expression.

▪ **Context-free languages.** The set of valid expressions in a context-free language can be defined recursively by rules such as *"e" → "e + e"* and *"e" → "(e)"* that specify how one expression can be built up from sequences of literal objects or "tokens" and other expressions. (As discussed on page 939, the fact that the left-hand side contains nothing more than *"e"* is what makes the language context free.) To interpret or parse an expression in a context-free language one has to go backwards and find out which rules could be used to generate that expression. (For the built-in syntax of *Mathematica* this is achieved using *ToExpression*.)

It is convenient to think of expressions in a language as having forms such as *s["(", "(", ")", ")"]* with *Attributes[s] = Flat*. Then the rules for the language consisting of balanced runs of parentheses (see page 939) can be written as

```
{s[e] → s[e, e], s[e] → s["(", e, ")"], s[e] → s["(", ")"]}
```

Different expressions in the language can be obtained by applying different sequences of these rules, say using (this gives so-called leftmost derivations)

```
Fold[#1 /. rules[[#2]] &, s[e], list]
```

Given an expression, one can then use the following to find a list of rules that will generate it—if this exists:

```
Parse[rules_, expr_] := Catch[Block[{t = {}}, NestWhile[
      ReplaceList[#, MapIndexed[ReverseRule, rules]] &,
      {{expr, {}}}, # /. {s[e], u_} :→ Throw[u]; # =!= {} &];]]
ReverseRule[a_ → b_, {i_}] := {___, {s[x___, b, y___], {u___}},
    ___} :→ {s[x, a, y], {i, u}} /; FreeQ[s[x], s[a]]
```

In general, there will in principle be more than one such list, and to pick the appropriate list in a practical situation one normally takes the rules of the language to apply with a certain precedence—which is how, for example, $x + y z$ comes to be interpreted in *Mathematica* as *Plus[x, Times[y, z]]* rather than *Times[Plus[x, y], z]*. (Note that in practice the output from a parser for a context-free language is usually represented as a tree—as in *Mathematica FullForm*—with each node corresponding to one rule application.)

Given only the rules for a context-free language, it is often very difficult to find out the properties of the language (compare page 944). Indeed, determining even whether two sets of rules ultimately yield the same set of expressions is in general undecidable (see page 1138).

▪ **Languages.** There are about 140 human languages and 15 full-fledged computer languages currently in use by a million people or more. Human languages typically have perhaps 50,000 reasonably common words; computer languages usually have a few hundred at most (*Mathematica*, however, has at least nominally somewhat over 1000). In expressing general human issues, different human languages tend to be largely equivalent—though they often differ when it comes to matters of special cultural or environmental interest to their users. Computer languages are also mostly equivalent in their handling of general programming issues—and indeed among widespread languages the only substantial exception is *Mathematica*, which supports symbolic, functional and pattern-based as well as procedural programming. Human languages have mostly evolved quite haphazardly over the course of many centuries, becoming sometimes simpler, sometimes more complicated. Computer languages are almost always specifically designed once and for all, usually by a single person. New human languages have sometimes been developed—a notable example being Esperanto in the 1890s—but for reasons largely of political history none have in practice become widely used.

Human languages always seem to have fairly definite rules for what is grammatically correct. And in a first approximation these rules can usually be thought of as specifying that every sentence must be constructed from various independent nested phrases, much as in a context-free grammar (see above). But in any given language there are always many exceptions, and in the end it has proved essentially impossible to identify specific detailed features—beyond for example the existence of nouns and verbs—that are convincingly universal across more than just languages with clear historical connections (such as the Indo-European ones). (One obvious general deviation from the context-free

model is that in practice subordinate clauses can never be nested too deep if a sentence is expected to be understood.)

All the computer languages that are in widespread use today are based quite explicitly on context-free grammars. And even though the original motivation for this was typically ease of specification or implementation, I strongly suspect that it has also been critical in making it readily possible for people to learn such languages. For in my observation, exceptions to the context-free model are often what confuse users of computer languages the most—even when those users have never been exposed to computer languages before. And indeed the same seems to be true for traditional mathematical notation, where occasional deviations from the context-free model in fields like logic seem to make material particularly hard to read. (A notable feature that I was surprised to discover in designing *Mathematica* 3 is that users of mathematical notation seem to have a remarkably universal view of the precedence of different mathematical operators.)

The idea of describing languages by grammars dates back to antiquity (see page 875). And starting in the 1800s extensive studies were made of the comparative grammars of different languages. But the notion that grammars could be thought of like programs for generating languages did not emerge with clarity until the work of Noam Chomsky beginning in 1956. And following this, there were for a while many efforts to formulate precise models for human languages, and to relate these to properties of the brain. But by the 1980s it became clear—notably through the failure of attempts to automate natural language understanding and translation—that language cannot in most cases (with the possible exception of grammar-checking software) meaningfully be isolated from other aspects of human thinking.

Computer languages emerged in the early 1950s as higher-level alternatives to programming directly in machine code. FORTRAN was developed in 1954 with a syntax intended as a simple idealization of mathematical notation. And in 1958, as part of the ALGOL project, John Backus used the idea of production systems from mathematical logic (see page 1150) to set up a recursive specification equivalent to a context-free grammar. A few deviations from this approach were tried—notably in LISP and APL—but by the 1970s, following the development of automated compiler generators such as yacc, so-called Backus-Naur context-free specifications for computer languages had become quite standard. (A practical enhancement to this was the introduction of two-dimensional grammar in *Mathematica* 3 in 1996.)

▪ **Page 631 · Computer language fluency.** It is common that when one knows a human language sufficiently well, one feels that one can readily "think in that language". In my experience the same is eventually true with computer languages. In particular, after many years of using *Mathematica*, I have now got to the point where I can effectively think directly in *Mathematica,* so that I can start entering a *Mathematica* program even though I may be a long way from being able to explain in English what I want to do.

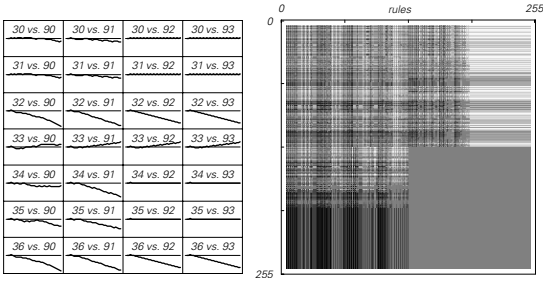
▪ **Brainteasers.** In many puzzles and IQ tests the setup is to give a few elements in some sequence of numbers, strings or pictures, then to ask what the next element would be. The correct answer is normally assumed to be the one that in a sense allows the simplest description of all the data. But despite attempts to remove cultural and other biases such questions in practice seem almost always to rely on being able to retrieve from memory various specific forms and transformations. And I strongly suspect that if one were, for example, to construct similar questions using outputs from many of the simple programs I discuss in this book then unless one had studied almost exactly the cases of such programs used one would never manage to work out the answers.

▪ **Human generation of randomness.** If asked to type a random sequence of 0's and 1's, most people will at first produce a sequence with too many alternations between 0 and 1. But with modest learning time my experience is that one can generate sequences with quite good randomness.

▪ **Game theory.** Remarkably simple models are often believed to capture features of what might seem like sophisticated decision making by humans, animals and human organizations. A particular case on which many studies have been done is the so-called iterated Prisoner's Dilemma, in which two players make a sequence of choices *a* and *b* to "cooperate" (*1*) or "defect" (*2*), each trying to maximize their score `m[[a, b]]` with `m = {{1, -1}, {2, 0}}`. At a single step, standard static game theory from the 1940s implies that a player should always defect, but in the 1960s a folk theorem emerged that if a whole sequence of steps is considered then a possible strategy for perfectly rational players is always to cooperate—in apparent agreement with some observations on human and animal behavior. In 1979 Robert Axelrod tried setting up computer programs as players and found that in tournaments between them the winner was often a simple "tit-for-tat" program that cooperates on the first step, then on subsequent steps just does whatever its opponent did on the previous step. The same winner was also often obtained by natural selection—a fact widely taken to explain cooperation phenomena in evolutionary biology and the social sciences. In the late 1980s similar studies were done on processes such as

auctions (cf page 1015), and in the late 1990s on games such as Rock, Paper, Scissors (RoShamBo) (with $m = \{\{0, -1, 1\}, \{1, 0, -1\}, \{-1, 1, 0\}\}$). (A simpler game—certainly played since antiquity—is Penny Matching or Evens and Odds, with $m = \{\{1, -1\}, \{-1, 1\}\}$.) But even though they seemed to capture or better actual human behavior, the programs considered in all these cases typically just used standard statistical or Markov model methods, or matching of specific sequences—making them far too weak to make predictions about the kinds of complex behavior shown in this book. (Note that a program can always win the games above if it can in effect successfully predict each move its opponent will make. In a game between two arbitrary programs it can be undecidable which will win more often over the course of an infinite number of moves.)

■ **Games between programs.** One can set up a game between two programs generating single bits of output by for example taking the input at each step to be the concatenation of the historical sequences of outputs from the two programs. The pictures below show what happens if the programs operate by applying elementary cellular automaton rules t times to $2t + 1$ inputs. The plots on the left show cumulative scores in the Evens and Odds game; the array on the right indicates for each of the 256 possible rules the average number of wins it gets against each of the 256 rules. At some level considerable complexity is evident. But the rules that win most often typically seem to do so in rather simple ways.



Higher Forms of Perception and Analysis

■ **Biological perception.** Animals can process data not only from visual or auditory sources (as discussed on pages 577 and 585), but also from mechanical, thermal, chemical and other sources. Usually special receptors for each type of data convert it into electrical impulses in nerve cells. Mechanical and thermal data are often mapped onto an array of nerve cells in the brain, from which features are extracted similar to those in visual perception. Taste involves data from solids and liquids; smell data from gases. The human tongue has millions of taste buds scattered on its surface, each with many tens of nerve cells. Rather little is currently known about how taste data is processed, and it is not even clear whether the traditional notion that there are just four or so primary tastes is correct. The human nose has several tens of millions of receptors, apparently broken into a few hundred distinct types. Each of these types probably has proteins that form pockets with definite shapes, making it respond to molecules whose shapes fit into these pockets. People typically distinguish a few thousand odors, presumably by comparing responses of different receptor types. (Foods usually contain tens of distinct odors; manufactured scents hundreds.) There is evidence that at the first level of processing in the brain all receptors of a given type excite nerve cells that lie in the same spatial region. But just how different regions are laid out is not clear, and may well differ between individuals. Polymers whose lengths differ by more than one or two repeating units often seem to smell different, and it is conceivable that elaborate general features of shapes of molecules can be perceived. But more likely there is no way to build up sophisticated taste or smell data—and no analog of any properties such as repetition or nesting.

■ **Page 634 · Evolving to predict.** If one thinks that biological evolution is infinitely powerful one might imagine that by emulating it one would always be able to find ways to predict any sequence of data. But in practice methods based, for example, on genetic programming seem to do at best only about as well as all sorts of other methods discussed in this chapter. And typically what limits them seems to be much the same as I argue in Chapter 8 limits actual biological evolution: that incremental changes are difficult to make except when the behavior is fairly simple. (See also page 985.)

It is common for animals to move in apparently random ways when they are trying to avoid predators. Yet I suspect that the randomness they use is often generated by quite simple rules (see page 1011)—so that in principle it could be predictable. So it is then notable that biological evolution has apparently never made predators able to catch their prey by predicting anything that looks to us particularly random; instead strategies tend to be based on tricks that do not require predicting more than at most repetition.

■ **Page 635 · Familiar features.** What makes features familiar to us is that they are common in our typical environment and are readily recognized by our built-in human powers of perception. In the distant past humans were presumably exposed only to features generated by ordinary natural

processes. But ever since the dawn of civilization humans have increasingly been exposed to things that were explicitly constructed through engineering, architecture, art, mathematics and other human activities. And indeed as human knowledge and culture have progressed, humans have ended up being exposed to new kinds of features. For example, while repetition has been much emphasized for several millennia, it is only in the past couple of decades that precise nesting has had much emphasis. So this may make one wonder what features will be emphasized in the future. The vast majority of forms created by humans in the past—say in art or architecture—have had basic features that are either directly copied from systems in nature, or are in effect built up by using extremely simple kinds of rules. On the basis of the discoveries in this book I thus tend to suspect that almost any feature that might end up becoming emphasized in the future will already be present—and probably even be fairly common—in the behavior of the kinds of simple programs that I have discussed in this book. (When future technology is routinely able to interact with individual atoms there will presumably quickly be a new class of quantum and other features that become familiar.)

■ **Relativism and postmodernism.** See pages 1131 and 1196.

The Notion of Computation

Computation as a Framework

▪ **History of computing.** Even in prehistoric times there were no doubt schemes for computation based for example on making specific arrangements of pebbles. Such schemes were somewhat formalized a few thousand years ago with the invention of the abacus. And by about 200 BC the development of gears had made it possible to create devices (such as the Antikythera device from perhaps around 90 BC) in which the positions of wheels would correspond to positions of astronomical objects. By about 100 AD Hero had described an odometer-like device that could be driven automatically and could effectively count in digital form. But it was not until the 1600s that mechanical devices for digital computation appear to have actually been built. Around 1621 Wilhelm Schickard probably built a machine based on gears for doing simplified multiplications involved in Johannes Kepler's calculations of the orbit of the Moon. But much more widely known were the machines built in the 1640s by Blaise Pascal for doing addition on numbers with five or so digits and in the 1670s by Gottfried Leibniz for doing multiplication, division and square roots. At first, these machines were viewed mainly as curiosities. But as the technology improved, they gradually began to find practical applications. In the mid-1800s, for example, following the ideas of Charles Babbage, so-called difference engines were used to automatically compute and print tables of values of polynomials. And from the late 1800s until about 1970 mechanical calculators were in very widespread use. (In addition, starting with Stanley Jevons in 1869, a few machines were constructed for evaluating logic expressions, though they were viewed almost entirely as curiosities.)

In parallel with the development of devices for digital computation, various so-called analog computers were also built that used continuous physical processes to in effect perform computations. In 1876 William Thomson (Kelvin) constructed a so-called harmonic analyzer, in which an assembly of disks were used to sum trigonometric series and thus to predict tides. Kelvin mentioned that a similar device could be built to solve differential equations. This idea was independently developed by Vannevar Bush, who built the first mechanical so-called differential analyzer in the late 1920s. And in the 1930s, electrical analog computers began to be produced, and in fact they remained in widespread use for finding approximate solutions to differential equations until the late 1960s.

The types of machines discussed so far all have the feature that they have to be physically rearranged or rewired in order to perform different calculations. But the idea of a programmable machine already emerged around 1800, first with player pianos, and then with Marie Jacquard's invention of an automatic loom which used punched cards to determine its weaving patterns. And in the 1830s, Charles Babbage described what he called an analytical engine, which, if built, would have been able to perform sequences of arithmetic operations under punched card control. Starting at the end of the 1800s tabulating machines based on punched cards became widely used for commercial and government data processing. Initially, these machines were purely mechanical, but by the 1930s, most were electromechanical, and had units for carrying out basic arithmetic operations. The Harvard Mark I computer (proposed by Howard Aiken in 1937 and completed in 1944) consisted of many such units hooked together so as to perform scientific calculations. Following work by John Atanasoff around 1940, electronic machines with similar architectures started to be built. The first large-scale such system was the ENIAC, built between 1943 and 1946. The focus of the ENIAC was on numerical computation, originally for creating ballistics tables. But in the early 1940s, the British wartime cryptanalysis group (which included Alan Turing) constructed fairly large electromechanical machines that performed logical, rather than arithmetic, operations.

All the systems mentioned so far had the feature that they performed operations in what was essentially a fixed sequence. But by the late 1940s it had become clear, particularly through the writings of John von Neumann, that it would be convenient to be able to jump around instead of always having to follow a fixed sequence. And with the idea of storing programs electronically, this became fairly easy to do, so that by 1950 more than ten stored-program computers had been built in the U.S. and in England. Speed and memory capacity have increased immensely since the 1950s, particularly as a result of the development of semiconductor chip technology, but in many respects the basic hardware architecture of computers has remained very much the same.

Major changes have, however, occurred in software. In the late 1950s and early 1960s, the main innovation was the development of computer languages such as FORTRAN, COBOL and BASIC. These languages allowed programs to be specified in a somewhat abstract way, independent of the precise details of the hardware architecture of the computer. But the languages were primarily intended only for specifying numerical calculations. In the late 1960s and early 1970s, there developed the notion of operating systems—programs whose purpose was to control the resources of a computer—and with them came languages such as C. And then in the late 1970s and early 1980s, as the cost of computer memory fell, it began to be feasible to manipulate not just purely numerical data, but also data representing text and later pictures. With the advent of personal computers in the early 1980s, interactive computing became common, and as the resolution of computer displays increased, concepts such as graphical user interfaces developed. In more recent years continuing increases in speed have made it possible for more and more layers of software to be constructed, and for many operations previously done with special hardware to be implemented purely in software.

▪ **Practical computers.** At the lowest level the hardware of a practical computer consists of digital electronic circuits. In these circuits, lumps of electric charge (in 2001 about half a million electrons each) flow through channels which cross to form various kinds of gates. Each gate performs a simple logic operation; for example, letting charge pass in one channel only if charge is present in the other channel. From circuits containing millions of such gates are built the two main elements of the computer: the processor which actually performs computations, and the memory which stores data. The memory consists of an array of cells, with the presence or absence of a lump of charge at gates in each cell representing a 1 or 0 value for the bit of data associated with that cell.

One of the crucial ideas of a general-purpose computer is that sequences of such bits of data in memory can represent information of absolutely any kind. Numbers for example are typically represented in base 2 by sequences of 32 or more bits. Similarly, characters of text are usually represented by sequences of 8 or more bits. (The character "a" is typically 01100001.) Images are usually represented by bitmaps containing thousands or millions of bits, with each bit specifying for example whether a pixel at a particular location should, say, be black or white. Every possible location in memory has a definite address, independent of its contents. The address is typically represented as a number which itself can be stored in memory.

What makes possible essential universality in a practical computer is that the data which is stored in memory can be a program. At the lowest level, a program consists of a sequence of instructions to be executed by the processor. Any particular kind of processor is built to support a certain fixed set of possible kinds of instructions, each represented by a specific number or opcode. There are typically a few tens of possible instructions, each executed by a certain part of the circuit in the processor. A typical one of these instructions might add two numbers together; a program would specify which numbers to add by giving their addresses in memory.

What practical computers always basically do is to repeat millions of times a second a simple cycle, in which the processor fetches an instruction from memory, then executes the instruction. The address of the instruction to be fetched at each point is specified by the current value of the program counter—a number stored in memory that is incremented by the processor, or can be modified by instruction in the program. At any given time, there are usually several programs stored in the memory of a computer, all organized by an operating system program which determines when other programs should run. Devices like keyboards, mice and microphones convert input into data that is inserted into memory at certain fixed locations. The operating system periodically checks these locations, and if necessary runs programs to respond to the input that is given.

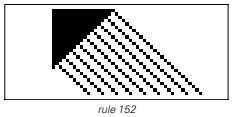
A crucial achievement in practical computing over the past several decades has been the creation of more and more sophisticated software. Often the programs that make up this software are several million instructions long. They usually contain many subprograms that perform parts of their task. Some programs are set up to perform very specific applications, say word processing. But an important class of programs are languages. A language provides a fixed set of constructs that allow one to specify computations. The set of instructions performed by the

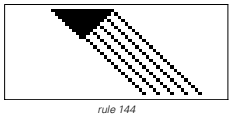
processor in a computer constitutes a low-level "machine" language. In practice, however, programs are rarely written at such a low level. More often, languages like C, FORTRAN, Java or *Mathematica* are used. In these languages, each construct represents what is often a large number of machine instructions. There are two basic ways that languages can operate: compiled or interpreted. In a compiled language like C or FORTRAN, the source code of the program must always first be translated by a compiler program into object code that essentially consists of machine instructions. Once compiled, a program can be executed any number of times. In an interpreted language, each piece of input effectively causes a fixed subprogram to be executed to perform an operation specified by that input.

▪ **Intuition from practical computing.** See page 872.

Computations in Cellular Automata

▪ **Page 639 · Other examples.** Rule 152 and rule 144, which effectively compute *Ceiling[n/2]* and *Ceiling[n/4]*, respectively, are shown below with *n = 18* initial black cells.


rule 152


rule 144

As discussed on page 989 rule 184 effectively determines whether its initial conditions correspond to a balanced sequence of open and close parentheses. (Rule 132 can be viewed as being like a syntax checker for a regular language; rule 184 for a context-free language.)

▪ **Page 639 · Squaring cellular automaton.** The rules are

```
{{0, _, 3}→0, {_, 2, 3}→3, {1, 1, 3}→4, {_, 1, 4}→4, {1|2, 3,
    _}→5, {p:(0|1), 4, _}→7-p, {7, 2, 6}→3, {7, _, _}→7,
  {_, 7, p:(1|2)}→p, {_, p:(5|6), _}→7-p, {5|6, p:(1|2), _}→
    7-p, {5|6, 0, 0}→1, {_, p:(1|2), _}→p, {_, _, _}→0}
```

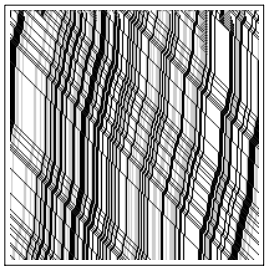
and the initial conditions consist of *Append[Table[1, {n}], 3]* surrounded by *0*'s. The rules can be implemented using *GeneralCARule* as given on page 867. (See also page 1186.)

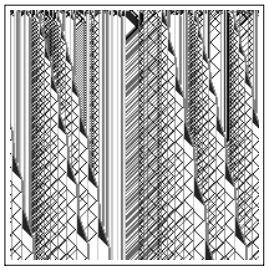
▪ **Page 640 · Primes cellular automaton.** The rules are

```
{{13, 3, 13}→12, {6, _, 4}→15, {10, _, 3|11}→15, {13, 7, _}→
    8, {13, 8, 7}→13, {15, 8, _}→1, {8, _, _}→7, {15, 1, _}→2,
  {_, 1, _}→1, {1, _, _}→8, {2|4|5, _, _}→13, {15, 2, _}→4,
  {_, 4, 8}→4, {_, 4, _}→5, {_, 5, _}→3, {15, 3, _}→12,
  {_, x:(2|3|8), _}→x, {_, x:(11|12), _}→x-1, {11, _, _}→13,
  {13, _, 1|2|3|5|6|10|11}→15, {13, 0, 8}→15,
  {14, _, 6|10}→15, {10, 0|9|13, 6|10}→15, {6, _, 6}→0,
  {_, _, 10}→9, {6|10, 15, 9}→14, {_, 6|10, 9|14|15}→10,
  {_, 6|10, _}→6, {6|10, 15, _}→13, {13|14, _, 9|15}→14,
  {13|14, _, _}→13, {_, _, 15}→15, {_, _, 9|14}→9, {_, _, _}→0}
```

and the initial conditions consist of *{10, 0, 4, 8}* surrounded by *0*'s. The right-hand region in the pattern grows like $\sqrt{t}$. (See also page 132.)

▪ **Random initial conditions.** The pictures below show the squaring and primes cellular automata starting from random initial conditions. Note that for both systems the majority of cases in their rules are not used in the specific computations for which they were constructed. Changing these cases can lead to different behavior with random initial conditions.





▪ **Efficiency of computations.** Present-day practical computers almost always process data in a basically sequential manner. Cellular automata, however, intrinsically operate in parallel, and can thus presumably perform at least some computations in fundamentally fewer steps. (Compare the discussion of P completeness on page 1149.)

▪ **Minimal programs for sequences.** See page 1186.

The Phenomenon of Universality

▪ **History of universality.** In Greek times it was noted as a philosophical matter that any single human language can be used to describe the same basic range of facts and processes. And with logic introduced as a way to formalize arguments (see page 1099), Gottfried Leibniz in the 1600s considered the idea of setting up a universal language based on logic that would provide a precise description analogous to a mathematical proof of any fact or process. But while Leibniz considered the possibility of checking his descriptions by machine, he apparently did not imagine setting up the analog of a computation in which something is explicitly generated from input that has been given.

The idea of having an abstract procedure that can be fed a range of different inputs had precursors in antiquity in the use of letters to denote objects in geometrical constructions, and in the 1500s in the introduction of symbolic formulas and algebraic variables. But the notion of abstract functions

in mathematics reached its modern form only near the end of the 1800s.

At the beginning of the 1800s practical devices such as the player pianos and the Jacquard loom were invented that could in effect be fed different inputs using analogs of punched cards. And in the 1830s Charles Babbage and Ada Lovelace noted that a similar approach could be used to specify the mathematical procedure to be followed by a mechanical calculating machine (see page 1107). But it was somehow assumed that the specification of the procedure must be done quite separately from the specification of the data to which the procedure was to be applied.

Starting in the 1880s attempts to build up both numbers and the operations of arithmetic from logic and set theory began to suggest that both data and procedures could potentially be described in common terms. And in the 1920s work by Moses Schönfinkel on combinators and by Emil Post on string rewriting systems provided fairly concrete examples of this.

In 1930 Kurt Gödel used the same basic idea to set up Gödel numbers to encode logical and other procedures as numbers. (Leibniz had in fact already done this for basic logic expressions in 1679.) But Gödel then took the crucial step of showing that the process of finding outputs from all such procedures could in effect be viewed as equivalent to following relations of logic and arithmetic—thus establishing that these relations are in a certain sense universal (see page 784). This fact, however, was embedded inside the rather technical proof of Gödel's Theorem, and it was at first not at all clear how specific it might be to the particular mathematical systems considered.

But in 1935 Alonzo Church constructed a system in lambda calculus that he showed could be made to emulate any other system in lambda calculus if given appropriate input, and in 1936 Alan Turing did the same thing for Turing machines. As discussed on page 1125, both Church and Turing argued that the systems they set up would be able to perform any reasonable computation. In both cases, their original motivation was to use this fact to construct an argument that the so-called decision problem (Entscheidungsproblem) of mathematical logic was undecidable (see page 1136). But Turing in particular gradually realized that his notion of universality could be applied to practical computers.

Turing's results were used in the 1940s—notably in the work of Warren McCulloch and Walter Pitts—as a basis for the assertion that electric circuit analogs of neural networks could achieve the sophistication of brains, and this appears to have influenced John von Neumann's thinking about the general programmability of electronic computers. Nevertheless, by the late 1940s, practical computer engineering had also been led to the idea of storing programs—like data—electronically, and in the 1950s it became widely understood that general-purpose practical computers could be viewed as universal systems.

Many theoretical investigations of universality were made in the 1950s and 1960s, but gradually the emphasis shifted more towards issues of languages and algorithms.

■ **Universality in *Mathematica*.** As an example of how different primitive operations can be used to do the same computation, the following are a few ways that the factorial function can be defined in *Mathematica*:

```
f[n_] := n!
f[n_] := n f[n - 1]; f[1] = 1
f[n_] := Product[i, {i, n}]
f[n_] := Module[{t = 1}, Do[t = t i, {i, n}]; t]
f[n_] := Module[{t = 1, i}, For[i = 1, i ≤ n, i++, t *= i]; t]
f[n_] := Apply[Times, Range[n]]
f[n_] := Fold[Times, 1, Range[n]]
f[n_] := If[n == 1, 1, n f[n - 1]]
f[n_] := Fold[#2[#1] &, 1, Array[Function[t, #1 t] &, n]]
f = If[#1 == 1, 1, #1 #0[#1 - 1]] &
```

A Universal Cellular Automaton

■ **Page 648 · Universal cellular automaton.** The rules for the universal cellular automaton are

```
{{_, 3, 7, 18, _} → 12, {_, 5, 7 | 8, 0, _} → 12, {_, 3, 10, 18, _} → 16,
 {_, 5, 10 | 11, 0, _} → 16, {_, 5, 8, 18, _} → 7, {_, 5, 14, 0 | 18, _} →
  12, {_, _, 8, 5, _} → 7, {_, _, 14, 5, _} → 12, {_, 5, 11, 18, _} → 10,
 {_, 5, 17, 0 | 18, _} → 16, {_, _, x : (11 | 17), 5, _} → x - 1,
 {_, 0 | 9 | 18, x : (7 | 10 | 16), 3, _} → x + 1, {_, 0 | 9 | 18, 12, 3, _} →
  14, {_, _, 0 | 9 | 18, 7 | 10 | 12 | 16, x : (3 | 5)} → 8 - x,
 {_, _, _, 8 | 11 | 14 | 17, x : (3 | 5)} → 8 - x, {_, 13, 4, _, x : (0 | 18)} →
  x, {18, _, 4, _, _} → 18, {_, _, 18, _, 4} → 18, {0, _, 4, _, _} → 0,
 {_, _, 0, _, 4} → 0, {4, _, 0 | 18, 1, _} → 3, {4, _, _, _, _} → 4,
 {_, _, 4, _, _} → 9, {_, 4, 12, _, _} → 7, {_, 4, 16, _, _} → 10,
 {x : (0 | 18), _, 6, _, _} → x, {_, 2, 6, 15, x : (0 | 18)} → x, {_, 12 | 16,
  6, 7, _} → 0, {_, 12 | 16, 6, 10, _} → 18, {_, 9, 10, 6, _} → 16,
 {_, 9, 7, 6, _} → 12, {9, 15, 6, 7, 9} → 0, {9, 15, 6, 10, 9} → 18,
 {9, _, 6, _, _} → 9, {_, 6, 7, 9, 12 | 16} → 12, {_, 6, 10, 9, 12 | 16} →
  16, {12 | 16, 6, 7, 9, _} → 12, {12 | 16, 6, 10, 9, _} → 16,
 {6, 13, _, _, _} → 9, {6, _, _, _, _} → 6, {_, _, 9, 13, 3} → 9,
 {_, 9, 13, 3, _} → 15, {_, _, _, 15, 3} → 3, {_, 3, 15, 0 | 18, _} → 13,
 {_, 13, 3, _, 0 | 18} → 6, {x : (0 | 18), 15, 9, _, _} → x,
 {_, 6, 13, _, _} → 15, {_, 4, 15, _, _} → 13, {_, _, _, 15, 6} → 6,
 {_, _, 2, 6, 15} → 1, {_, _, 1, 6, _} → 2, {_, 1, 6, _, _} → 9, {_, 3, 2,
  _, _} → 1, {3, 2, _, _, _} → 3, {_, _, 3, 2, _} → 3, {_, 1, 9, 1, 6} → 6,
 {_, _, 9, 1, 6} → 4, {_, 4, 2, _, _} → 1, {_, _, _, _, x : (3 | 5)} → x,
 {_, _, 3 | 5, _, x : (0 | 18)} → x, {_, _, x : (1 | 2 | 7 | 8 | 9 | 10 | 11 |
     12 | 13 | 14 | 15 | 16 | 17), _, _} → x, {_, _, 18, 7 | 10, 18} → 18,
 {_, _, 0, 7 | 10, 0} → 0, {_, _, 0 | 18, _, _} → 9, {_, _, x_, _, _} → x}
```

where the numbers correspond to the icons shown in the main text according to

The block in the initial conditions for the universal cellular automaton corresponding to a cell with color a is given by

```
Flatten[{Transpose[{Join[{4, 18 (1 - a), 6}, Table[9,
    {2^(2 r+1) - 3}]], 10 - 3 rtab}], Table[{9, 1}, {r}], 9, 13}]
```

where r is the range of the rule to be emulated ($r = 1$ for elementary rules) and *rtab* is the list of outcomes for that rule (starting with the outcome for *{1, 1, (1) ...}*). In general, there are 2^{2r+1} cases in the rule to be emulated; each block in the universal cellular automaton is $2(2^{2r+1} + r + 1)$ cells wide, and each step in the rule to be emulated corresponds to $(3r + 2)2^{2r+1} + 3r^2 + 7r + 3$ steps in the evolution of the universal cellular automaton.

▪ **Page 655 · More colors.** Given a rule that involves three colors and nearest neighbors, the following converts each case of the rule to a collection of cases for a rule with two colors:

```
CA3ToCA2[{a_, b_, c_} → d_] := Union[Flatten[Table[Thread[
        Partition[Flatten[{l, a, b, c, r} /. coding], 11, 1][[{2,
          3, 4}]] → (d /. coding)], {l, 0, 2}, {r, 0, 2}], 2]]
coding = {0 → {0, 0, 0}, 1 → {0, 0, 1}, 2 → {0, 1, 1}}
```

The problem of encoding cells with several colors by blocks of black and white cells is related to standard problems in coding theory (see page 560). One approach is to use *{1, 1}* to indicate the boundary of each block, and then within each block to use all possible digit sequences which do not contain *{1, 1}*, as in the Fibonacci number system discussed on page 892. Note that the original rule with k colors and r neighbors involves $Log[2, k^{2r+1}]$ bits of information; the two-color rule that emulates it involves $Log[2, 2^{2^{2s+1}}]$ bits. As a result, the minimum possible s for $k = 3$, $r = 1$ is about 2.2; in the specific example shown in the main text it is 5.

Emulating Other Systems with Cellular Automata

▪ **Page 657 · Mobile automata.** Given a mobile automaton with rules in the form used on page 887, a cellular automaton which emulates it can be constructed using

```
MAToCA[rules_] :=
  Append[Flatten[Map[g, rules]], {_, _, x_, _, _} → x]
g[{a_, b_, c_} → {d_, e_}] := {{_, a, b + 2, c, _} → d, If[e == 1,
    {a, b + 2, c, _, _} → c + 2, {_, _, a, b + 2, c} → a + 2]}
```

This specific definition assumes that the mobile automaton has two possible colors for each cell; it yields a cellular automaton with four possible colors for each cell. An initial condition with a single 2 surrounded by 0's corresponds to all cells being white in the mobile automaton.

▪ **Page 658 · Turing machines.** Given any Turing machine with rules in the form used on page 888 and k possible colors for each cell, a cellular automaton which emulates it can be constructed using

```
TMToCA[rules_, k_ : 2] :=
  Flatten[{Map[g[#, k] &, rules], {_, x_, _} → x}]
g[{s_, a_} → {sp_, ap_, d_}, k_] := {If[d == 1, Identity,
    Reverse][{k s + a, x_, _}] → k sp + x, {_, k s + a, _} → ap}
```

If the Turing machine has s states for its head, then the cellular automaton has $k(s + 1)$ colors for each cell. An initial condition with a single cell of color k surrounded by 0's corresponds to being in state 1 with a blank tape in the Turing machine.

▪ **Page 659 · Substitution systems.** Given a substitution system with rules in the form such as *{1 → {0}, 0 → {0, 1}}* used on page 889, the rules for a cellular automaton which emulates it are obtained from

```
SSToCA[rules_] := {{b, b, p[x_, _]} → s[x],
  {_, s[v : (0 | 1)], p[x_, _]} → p[v, x], {_, p[_, y_], _} → s[y],
  {_, s[v : (0 | 1)], _m} → m[v], {s[0 | 1], m[v : (0 | 1)], _} →
    s[v], {b, m[v : (0 | 1)], _} → r[v], {_, r[v : (0 | 1)], _} :→
    (Replace[v, rules] /. {{x_} → s[x], {x_, y_} → p[x, y]}),
  {_r, s[v : (0 | 1)], _} → r[v], {_r, b, _} → m[b],
  {s[0 | 1], m[b], _} → b, {_, v_, _} → v}
```

where specific values for cells can be obtained from

```
{b → 0, s[0] → 1, m[0] → 2, p[0, 0] → 3,
  r[0] → 4, p[0, 1] → 5, p[1, 0] → 6, r[1] → 7,
  p[1, 1] → 8, m[1] → 9, m[b] → 10, s[1] → 11}
```

An initial condition consisting of a single element with color i in the substitution system is represented by *m[i]* surrounded by *b*'s in the cellular automaton. The specific definition given above works for neighbor-independent substitution systems whose elements have two possible colors, and in which each element is replaced at each step by at most two new elements.

▪ **Page 660 · Sequential substitution systems.** Given a sequential substitution system with rules in the form used on page 893, the rules for a cellular automaton which emulates it can be obtained from

```
SSSToCA[rules_] := Flatten[{{v[_, _, _], u, _} → u, {_, v[rn_,
  x_, _], u} → r[rn + 1, x], {_, v[_, x_, _], _} → x, MapIndexed[
  With[{rn = #2[[1]], rs = #1[[1]], rr = #1[[2]]}, {If[Length[rs] ==
    1, {u, r[rn, First[rs]], _} → q[0, rr], {u, r[rn, First[rs]], _} →
    v[rn, First[rs], Take[rs, 1]]], {u, r[rn, x_], _} → v[rn, x, {}],
  {v[rn, _, Drop[rs, -1]], Last[rs], _} → q[Length[rs] - 1, rr],
  Table[{v[rn, _, Flatten[{___, Take[rs, i - 1]}]], rs[[i]], _} → v[
    rn, rs[[i]], Take[rs, i]], {i, Length[rs] - 1, 1, -1}], {v[rn, _, _],
  y_, _} → v[rn, y, {}]}] &, rules /. s → List], {_, q[0, {x__, _}],
```

```
_} → q[0, {x}], {_, q[0, {x_}], _} → r[1, x], {_, q[0, {}], x_} →
 r[1, x], {_, q[_, {___, x_}], _} → x, {_, q[_, {}], x_} → x,
 {_, x_, q[0, _]} → x, {_, _, q[n_, {}]} → q[n - 1, {}],
 {_, _, q[n_, {x___, _}]} → q[n - 1, {x}], {q[_, {}], _, _} → w,
 {q[0, {__, x_}], p[y_, _], _} → p[x, y], {q[0, {__, x_}], y_, _} →
 p[x, y], {p[_, x_], p[y_, _], _} → p[x, y], {p[_, x_], u, _} → x,
 {p[_, x_], y_, _} → p[x, y], {_, p[x_, _], _} → x, {w, u, _} → u,
 {w, x_, _} → w, {_, w, x_} → x, {_, r[rn_, x_], _} → x,
 {_, u, r[_, _]} → u, {_, x_, r[rn_, _]} → r[rn, x], {_, x_, _} → x}]
```

The initial condition is obtained by applying the rule *s[x_, y__] → {r[1, x], y}* and then padding with *u*'s.

■ **Page 661 · Register machines.** Given the program for a register machine in the form used on page 896, the rules for a cellular automaton that emulates it can be obtained from

```
g[i[1], p_, m_] :=
  {{_, p, _} → p + 1, {_, 0, p} → m + 2, {_, _, p} → m + 3}
g[i[2], p_, m_] :=
  {{_, p, _} → p + 1, {p, 0, _} → m + 5, {p, _, _} → m + 6}
g[d[1, q_], p_, m_] := {{m + 2 | m + 3, p, _} → q, {m + 1,
    p, _} → p, {0, p, _} → p + 1, {_, m + 2 | m + 3, p} → m + 1}
g[d[2, q_], p_, m_] := {{_, p, m + 5 | m + 6} → q, {_, p,
    m + 4} → p, {_, p, 0} → p + 1, {p, m + 5 | m + 6, _} → m + 4}
RMToCA[prog_] := With[{m = Length[prog]}, Flatten[
  {MapIndexed[g[#1, First[#2], m] &, prog], {{0, 0 | m + 1,
      m + 3} → m + 2, {0, m + 1, _} → 0, {0, 0, m + 1} → 0,
    {_, _, x : (m + 1 | m + 3)} → x, {_, m + 1 | m + 3, _} → m + 2,
    {m + 6, 0 | m + 4, 0} → m + 5, {_, m + 4, 0} → 0,
    {m + 4, 0, 0} → 0, {x : (m + 4 | m + 6), _, _} → x,
    {_, m + 4 | m + 6, _} → m + 5, {_, x_, _} → x}}]]
```

If *m* is the length of the register machine program, then the resulting cellular automaton has *m + 7* possible colors for each cell. If the initial numbers in the two registers are *a* and *b*, then the initial conditions for the cellular automaton are *Join[Table[m + 2, {a}], {1}, Table[m + 5, {b}]]* surrounded by 0's.

■ **Page 661 · Multiplication systems.** The rules for the cellular automaton shown here are

```
{{_, 0, 3 | 8} → 5, {_, 0, 2 | 7} → 8, {_, 1, 4 | 9} → 9,
  {_, 1, 3 | 8} → 4, {_, 1, 2 | 7} → 8, {_, 10, 4 | 9} → 3,
  {_, 10, 3 | 8} → 7, {_, 10, 2 | 7} → 2, {5 | 6, 1, 0} → 9,
  {5 | 6, 10, 0} → 3, {5 | 6, 1, _} → 6, {5 | 6, 10, _} → 5,
  {_, 2 | 3 | 4 | 5, _} → 10, {_, 6 | 7 | 8 | 9, _} → 1, {_, x_, _} → x}
```

and the initial condition consists of a single *3* surrounded by *0*'s. The idea used is that multiplication by 3 can be achieved by scanning digits from right to left, adding to each digit the value of the digit on its immediate right, as well as a carry that can propagate any distance but cannot be larger than 1. Note that as discussed on page 614 multiplication by some multipliers in some bases (such as by 3 in base 6) can be achieved by a single step in the evolution of a suitable cellular automaton. After *t* steps, the width of the pattern shown here is about *Sqrt[Log[2, 3] t]*. (See also page 119.)

■ **Continuous systems.** See page 1128.

■ **Page 662 · Logic circuits.** The rules for the cellular automaton shown here are

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

The initial conditions are given by

```
Flatten[Block[{And, Or}, Map[{0, 2 (# + 1)} &, expr, {-1}] //.
    {! x_ :→ {0, x, 0}, And[x__] :→ {0, 0, 1, 0, x, 1, 3, 0, 0},
      Or[x__] :→ {0, 0, 1, 0, x, 0, 1, 3, 0}}]]
```

and in terms of these initial conditions the cellular automaton must be run for *Length[list //. {0, x__} → {x}] - 1* steps in order to find the result.

■ **Page 663 · RAM.** The rules for the cellular automaton shown here are

```
{{2, 4 | 8, 2 | 11, _, _} → 2, {11 | 10, 4 | 8, 2 | 11, _, _} → 11,
  {2, 4 | 8, _, _, _} → 10, {11 | 10, 4 | 8, _, _, _} → 2,
  {2, 0, _, _, _} → 2, {11, 0, _, _, _} → 11,
  {3 | 7 | 6, _, 10, _, _} → 1, {x : (3 | 7 | 6), _, _, _, _} → x,
  {_, _, 6, 4, 10} → 5, {_, _, 6, 8, 10} → 9, {_, 3, _, 10, _} → 4,
  {_, 7, _, 10, _} → 8, {_, _, 1, _, x : (5 | 9)} → x, {1, _, _, _, _} → 1,
  {_, _, 1, _, _} → 1, {_, _, _, _, 1} → 1, {_, _, x : (4 | 8 | 0), _, _} → x}
```

The initial conditions are divided into two parts: instructions on the left and memory on the right. Given a list of *0* and *1* values for successive memory locations, the right-hand initial conditions are *Flatten[list /. {1 → {8, 1}, 0 → {4, 1}}]*. To access location *n* the left-hand initial conditions must contain *Flatten[{0, i, IntegerDigits[n, 2] /. {1 → {0, 11}, 0 → {0, 2}}}]* inserted in a repetitive *{0, 1}* background. If *i* is *7*, a *1* will be written to location *n*; if it is *3*, a *0* will be written; and if it is *6*, the contents of location *n* will be read and sent back to the left.

Emulating Cellular Automata with Other Systems

■ **Page 664 · Mobile automata.** Given the rules for an elementary cellular automaton in the form used on page 867, the following will construct a mobile automaton which emulates it:

```
vals = {x, p[0], q[0, 0], q[0, 1], q[1, 0], q[1, 1], p[1]}

CAToMA[rules_] := Table[(# → Replace[#, {{q[a_, b_], p[c_],
      p[d_]} :→ {q[c, {a, c, d} /. rules], 1}, q[a_, b_], p[c_], x} :→
     {q[c, {a, c, 0} /. rules], 1}, {q[_, _], x, x} → {p[0], -1},
    {q[_, _], q[_, a_], p[_]} → {p[a], -1}, {x, q[_, a_], p[_]} →
     {p[a], -1}, {x, x, p[_]} → {q[0, 0], 1}, {_, _, _} →
     {x, 0}}] &)[vals[[IntegerDigits[i, 7, 3] + 1]]], {i, 0, 7^3 - 1}]
```

The ordering in *vals* defines a mapping of symbolic cell values onto colors. Given a list of initial cell colors for the cellular automaton, the initial conditions for the mobile automaton are given by *Flatten[{p[0], Map[p, list], p[0]}]* surrounded by *x*'s, with the active cell being placed initially just before the first *p[0]*.

■ **Page 665 · Turing machines.** Given the rules for an elementary cellular automaton in the form used on page 867, the following will construct a Turing machine which emulates it:

```
CAToTM[rules_] :=
 {{q[a_, b_], c : (0 | 1)} :> {q[b, c], {a, b, c} /. rules, 1},
  {q[_, _], x} → {p[0], 0, -1}, {p[a_], b : (0 | 1)} →
   {p[b], a, -1}, {p[_], x} → {q[0, 0], 0, 1}}
```

Given a list of initial cell colors for the cellular automaton, the initial tape for the Turing machine consists of *Join[{0, 0}, list, {0, 0}]* surrounded by *x*'s, with the head of the Turing machine on the first *0* in state *q[0, 0]*.

For specific cellular automata it is often possible to construct smaller Turing machines, as on pages 707 and 1119. By combining identical cases in rules and writing rules as compositions of ones with smaller neighborhoods one can for example readily construct Turing machines with 4 states and 3 colors that emulate 166 of the elementary cellular automata.

■ **Page 667 · Sequential substitution systems.** Given the rules for an elementary cellular automaton in the form used on page 867, the following will construct a sequential substitution system which emulates it:

```
CAToSSS[rules_] := Join[rules /.
      ({a_, b_, c_} → d_) → {1, 2 a, 2 b, 2 c} → {2 d, 1, 2 b, 2 c},
   {{1, 0, 0} → {0, 0}, {0} → {1, 0, 0, 0}}]
```

The initial condition *{0, 0, 2, 0, 0}* for the sequential substitution system corresponds to a single black cell surrounded by white cells in the cellular automaton.

■ **Page 667 · Tag systems.** Given the rules for an elementary cellular automaton in the form used on page 867, the following will construct a tag system which emulates it:

```
CAToTS[rules_] := {2, {{s[x_], s[y_]} :>
       {d[x, y], d[x, y]}, {d[w_, x_], d[y_, z_]} :>
       {s[{w, x, y} /. rules], s[{x, y, z} /. rules]},
    {s[x_], d[y_, z_]} :> {s[0], s[0], s[{0, y, z} /. rules]},
    {d[x_, y_], s[z_]} :> {s[{x, y, 0} /. rules], s[0], s[0]}}}
```

The initial condition for the tag system that corresponds to a single black cell in the cellular automaton is *{s[0], s[0], s[1], s[0], s[0]}*. Given a list of all steps in the evolution of the tag system, *Cases[list, {__s}]* picks out successive steps in the cellular automaton evolution.

■ **Page 668 · Symbolic systems.** Given the rules for an elementary cellular automaton in the form used on page 867 (with *{0, 0, 0} → 0*), the following will construct a symbolic system which emulates it:

```
Flatten[{Array[p[x_][#1][#2][#3] →
   p[x[{##} /. rules]][#2][#3] &, {2, 2, 2}, 0] /. {0 → p, 1 → q},
  {r[x_] → p[r[p][p]][x], p[x_][p][p][r] → x[p][p][r]}}]
```

The initial condition for the symbolic system is given by

```
Fold[#1[#2] &, r[p][p], init /. {0 → p, 1 → q}][p][p][r]
```

Step *t* in the cellular automaton corresponds to step *t (t + Length[init] + 3)* in the symbolic system.

Note that the succession of states shown here depends on the detailed order in which rules are applied (see page 898). It is also possible to construct symbolic systems with the so-called confluence property, in which results from any fixed number of steps of cellular automaton evolution can be found by applying rules in any possible order (see page 1036).

■ **Page 669 · Cyclic tag systems.** From a tag system which depends only on its first element, with rules given as in the note below, the following constructs a cyclic tag system emulating it:

```
TS1ToCT[{n_, subs_}] := With[{k = Length[subs]},
    Join[Map[v[Last[#], k] &, subs], Table[{}, {k (n - 1)}]]]
u[i_, k_] := Table[If[j == i + 1, 1, 0], {j, k}]
v[list_, k_] := Flatten[Map[u[#, k] &, list]]
```

The initial condition for the tag system can be converted using *v[list, k]*. The list representing the complete history of the resulting cyclic tag system can then be interpreted using

```
Map[Map[Position[#, 1][[1, 1]] - 1 &, Partition[#, k]] &,
  Take[history, {1, -1, n k}]]
```

This construction is relevant to the proof of the universality of rule 110 starting on page 678.

■ **Page 669 · Multicolor Turing machines.** Given rules in the form on page 888 for a Turing machine with *s* states and *k* colors the following yields an equivalent Turing machine with *With[{c = Ceiling[Log[2, k]]}, ((3 2^c) + 2 c - 7) s]* states (always less than *6.03 k s*) and 2 colors:

```
TMToTM2[rule_, s_, k_] := (# /. MapIndexed[
    #1 → First[#2] &, Union[Map[#[[1, 1]] &, #]]] &)[
  With[{b = Ceiling[Log[2, k]] - 1}, Flatten[Table[
    {Table[{Table[{{m, i, n, d}, c} → {{m, Mod[i, 2^(n-1)], n - 1,
         d}, Quotient[i, 2^(n-1)], 1}, {n, 2, b}, {i, 0, 2^n - 1}], Table[
       {{m, i, 1, d}, c} → {{m, -1, 1, d}, i, d}, {i, 0, 1}], Table[
       {{m, -1, n, d}, c} → {{m, -1, n + 1, d}, c, d}, {n, b - 1}],
       {{m, -1, b, d}, c} → {{0, 0, m}, c, d}}, {d, -1, 1, 2}],
     Table[{{i, n, m}, c} → {{i + 2^n c, n + 1, m}, c, -1},
      {n, 0, b - 1}, {i, 0, 2^n - 1}], With[{r = 2^b}, Table[
       If[i + r c ≥ k, {}, Cases[rule, ({m, i + r c} → {x_, y_, z_}) →
         {{i, b, m}, c} → {{x, Mod[y, r], b, z}, Quotient[y, r],
          1}]], {i, 0, r - 1}]]], {m, s}, {c, 0, 1}]]]]
```

Some of these states are usually unnecessary, and in the main text such states have been pruned. Given an initial condition *{i, list, n}* the initial condition for the 2-color Turing machine is

```
With[{b = Ceiling[Log[2, k]]},
  {i, Flatten[IntegerDigits[list, 2, b]], b n}]
```

■ **Page 670 · One-element-dependence tag systems.** Writing the rule *{3, {{0, _, _} → {0, 0}, {1, _, _} → {1, 1, 0, 1}}}* from page 895 as *{3, {0 → {0, 0}, 1 → {1, 1, 0, 1}}}* the evolution of a tag system that depends only on its first element is obtained from

```
TS1EvolveList[rule_, init_, t_] :=
  NestList[TS1Step[rule, #] &, init, t]

TS1Step[{n_, subs_}, {}] = {}

TS1Step[{n_, subs_}, list_] :=
  Drop[Join[list, First[list] /. subs], n]
```

Given a Turing machine in the form used on page 888 the following will construct a tag system that emulates it:

```
TMToTS1[rules_] :=
 {2, Union[Flatten[rules /. ({i_, u_} → {j_, v_, r_}) :>
    {Map[#[i] → {#[i, 1], #[i, 0]} &, {a, b, c, d}], If[r == 1,
     {a[i, u] → {a[j], a[j]}, b[i, u] → Table[b[j], {4}], c[i, u] →
      Flatten[{Table[b[j], {2 v}], Table[c[j], {2 - u}]}],
     d[i, u] → {d[j]}}, {a[i, u] → Table[a[j], {2 - u}],
     b[i, u] → {b[j]}, c[i, u] → Flatten[{c[j], c[j],
       Table[d[j], {2 v}]}], d[i, u] → Table[d[j], {4}]}]]]]}
```

A Turing machine in state *i* with a blank tape corresponds to initial condition *{a[i], a[i], c[i]}* for the tag system. The configuration of the tape on each side of the head in the Turing machine evolution can be obtained from the tag system evolution using

```
Cases[history, x : {a[_], ___} :>
    Apply[{#1, Reverse[#2]} &, Map[
        Drop[IntegerDigits[Count[x, #], 2], -1] &, {_b, _d}]]]
```

■ **Page 672 · Register machines.** Given the rules for a Turing machine in the form used on page 888, a register machine program to emulate the Turing machine can be obtained by techniques analogous to those used in compilers for practical computer languages. Here *TMCompile* creates a program segment for each element of the Turing machine rule, and *TMToRM* resolves addresses and links the segments together.

```
TMToRM[rules_] := Module[{segs, adrs}, segs =
  Map[TMCompile, rules]; adrs = Thread[Map[First, rules] →
    Drop[FoldList[Plus, 1, Map[Length, segs]], -1]];
  MapIndexed[#1 /. {dr[r_, n_] :> d[r, n + First[#2]],
    dm[r_, z_] :> d[r, z /. adrs]} &, Flatten[segs]]]

TMCompile[_ → z : {_, _, 1}] := f[z, {1, 2}]

TMCompile[_ → z : {_, _, -1}] := f[z, {2, 1}]

f[{s_, a_, _}, {ra_, rb_}] := Flatten[{i[3], dr[ra, -1],
    dr[3, 3], i[ra], i[ra], dr[3, -2], If[a == 1, i[ra], {}], i[3],
    dr[rb, 5], i[rb], dr[3, -1], dr[rb, 1], dm[rb, {s, 0}],
    dr[rb, -6], i[rb], dr[3, -1], dr[rb, 1], dm[rb, {s, 1}]}]
```

A blank initial tape for the Turing machine corresponds to initial conditions *{1, {0, 0, 0}}* for the register machine. (Assuming that the Turing machine starts in state 1, with a 0 under its head, other initial conditions can be encoded just by taking the values of cells on the left and right to give the digits of the numbers that are initially in the first two registers.) Given the list of successive configurations of the register machine, the steps that correspond to successive steps of Turing machine evolution can be obtained from

```
(Flatten[Partition[Complement[#, # - 1], 1, 2]] &)[
  Position[list, {_, {_, _, 0}}]]
```

The program given above works for Turing machines with any number of states, but it requires some simple extensions to handle more than two possible colors for each cell. Note that for a Turing machine with *s* states, the length of the register machine program generated is between *34 s* and *36 s*.

■ **Register machines with many registers.** It turns out that a register machine with any number of registers can always be emulated by a register machine with just two registers. The basic idea is to encode the list of values of all the registers in the multiregister machine in the single number given by

```
RMEncode[list_] :=
  Product[Prime[j]^list[[j]], {j, Length[list]}]
```

and then to have this number be the value at appropriate steps of the first register in the 2-register machine. The program in the multiregister machine can be converted to a program for the 2-register machine according to

```
RMToRM2[prog_] :=
  Module[{segs, adrs}, segs = MapIndexed[seg, prog];
    adrs = FoldList[Plus, 1, Map[Length, segs]];
    MapIndexed[#1 /. {ds[r_, s_] :> d[r, adrs[[s]]],
        dr[r_, j_] :> d[r, j + First[#2]]} &, Flatten[segs]]]

seg[i[r_], {a_}] := With[{p = Prime[r]},
    Flatten[{Table[i[2], {p}], dr[1, -p], i[1],
      dr[2, -1], Table[dr[1, 1], {p + 1}]}]]

seg[d[r_, n_], {a_}] := With[{p = Prime[r]}, Flatten[{i[2], dr[
  1, 5], i[1], dr[2, -1], dr[1, 1], ds[1, n], Table[{If[m == p - 1,
   ds[1, a], dr[1, 3 p + 2 - m]], Table[i[1], {p}], dr[2, -p],
   Table[dr[1, 1], {2 p - m - 1}], ds[1, a + 1]}, {m, p - 1}]}]]
```

The initial conditions for the 2-register machine are given by *{1, {RMEncode[list], 0}}* and the results corresponding to each step in the evolution of the multiregister machine appear whenever register 2 in the 2-register machine is incremented from 0.

■ **Computations with register machines.** As an example, the following program for a 3-register machine starting with initial condition *{n, 0, 0}* will compute *{Round[$\sqrt{n}$], 0, 0}*:

```
{d[1, 4], i[1], d[1, 15], i[2], d[1, 6], d[1, 11], i[1],
  d[2, 7], d[3, 7], d[1, 15], d[3, 4], i[3], d[2, 12], d[3, 4]}
```

■ **Page 673 · Arithmetic systems.** Given the program for a register machine with *nr* registers in the form on page 896, an arithmetic system which emulates it can be obtained from

```
RMToAS[prog_, nr_] := With[{p = Length[prog], g =
   Product[Prime[j], {j, nr}]}, {p g, Sort[Flatten[MapIndexed[
    With[{n = First[#2] - 1}, #1 /. {i[r_] :> Table[n + j p →
       (1 + n + Prime[r] (-n + #) &), {j, 0, g - 1}], d[r_, k_] :>
      Table[n + j p → If[Mod[j, Prime[r]] == 0, -1 + k + (-n +
         #)/Prime[r] &, # + 1 &], {j, 0, g - 1}]}] &, prog]]]}]
```

The rules for the arithmetic system are represented so that the system from page 122 becomes for example *{2, {0 :→ (3 #/2 &), 1 :→ (3 (# + 1)/2 &)}}*. If the register machine starts at instruction *n* with values *regs* in its registers, then the corresponding arithmetic system starts with the number *n + Table[Prime[i] ^ reg[[i]], {i, nr}] p - 1* where *p = Length[prog]*. The evolution of the arithmetic system is given by

```
ASEvolveList[{n_, rules_}, init_, t_] :=
  NestList[(Mod[#, n] /. rules)[#] &, init, t]
```

Given a value *m* obtained in the evolution of the arithmetic system, the state of the register machine to which it corresponds is

```
{Mod[m, p] + 1, Map[Last, FactorInteger[
      Product[Prime[i], {i, nr}] Quotient[m, p]]] - 1}
```

Note that it is possible to have each successive step involve only multiplication, with no addition, at the cost of using considerably larger numbers overall.

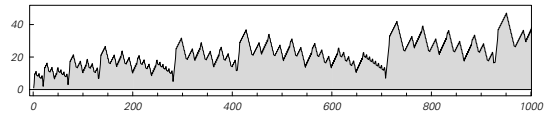
▪ **History.** The correspondence between arithmetic systems and register machines was established (using a slightly different approach) by Marvin Minsky in 1962. Additional work was done by John Conway, starting around 1971. Conway considered fraction systems based on rules of the form

```
FSEvolveList[fracs_, init_, t_] :=
  NestList[First[Select[fracs #, IntegerQ, 1]] &, init, t]
```

With the choice

```
fracs = {17/91, 78/85, 19/51, 23/38, 29/33, 77/29, 95/
    23, 77/19, 1/17, 11/13, 13/11, 15/14, 15/2, 55/1}
```

starting at *2* the result for *Log[2, list]* is as shown below, where *Rest[Log[2, Select[list, IntegerQ[Log[2, #]] &]]]* gives exactly the primes.



(Compare the discussion of universality in integer equations on page 786.)

▪ **Multiway systems.** It is straightforward to emulate a *k*-color multiway system with a 2-color one, just by encoding successive colors by strings like *"AAABBB"*, *"AAABAB"* and *"AABABB"* that have no overlaps. (Compare page 1033.)

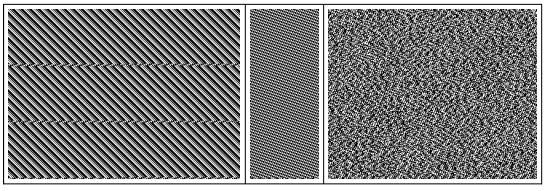
The Rule 110 Cellular Automaton

▪ **History.** The fact that 1D cellular automata can be universal was discussed by Alvy Ray Smith in 1970—who set up an 18-color nearest-neighbor cellular automaton rule capable of emulating Marvin Minsky's 7-state 4-color universal Turing machine (see page 706). (Roger Banks also mentioned in 1970 a 17-color cellular automaton that he believed was universal.) But without any particular reason to think it would be interesting, almost nothing was done on finding simpler universal 1D cellular automata. In 1984 I suggested that cellular automata showing what I called class 4 behavior should be universal—and I identified some simple rules (such as *k = 2*, *r = 2* totalistic code 20) as explicit candidates. A piece published in *Scientific American* in 1985 describing my interest in finding simple 1D universal cellular automata led me to receive a large number of proofs of the fact (already well known to me) that 1D cellular automata can in principle emulate Turing machines. In 1989 Kristian Lindgren and Mats Nordahl constructed a 7-color nearest-neighbor cellular automaton that could emulate Minsky's 7,4 universal Turing machine, and showed that in general a rule with *s + k + 2* colors could emulate an *s*-state *k*-color Turing machine (compare page 658). Following my ideas about class 4 cellular automata I had come by 1985 to suspect that rule 110 must be universal. And when I started working on the writing of this book in 1991, I decided to try to establish this for certain. The general outline of what had to be done was fairly clear—but there were an immense number of details to be handled, and I asked a young assistant of mine named Matthew Cook to investigate them. His initial results were encouraging, but after a few months he became increasingly convinced that rule 110 would never in fact be proved universal. I insisted, however, that he keep on trying, and over the next several years he developed a systematic computer-aided design system for working with structures in rule 110. Using this he was then in 1994 successfully able to find the main elements of the proof. Many details were filled in over the next year, some mistakes were corrected in 1998, and the specific version in the note below was constructed in 2001. Like most proofs of universality, the final proof he found is conceptually quite straightforward, but is filled with many excruciatingly elaborate details. And among these details it is certainly possible that a few errors still remain. But if so, I believe that they can be overcome by the same general methods that have been used in the proof so far. Quite probably a somewhat simpler proof can be given, but as discussed on page 722 it is essentially inevitable that proofs of universality must be at least somewhat complicated. In the future it should be possible to give a proof in a form that can be checked completely by computer. (The initial conditions in the note below quite soon become too large to run explicitly on any existing computer.) And in addition, with sufficient effort, I believe one should be able to construct an automated system that will allow many universality proofs of this general kind to be found almost entirely by computer (compare page 810).

■ **Page 683 · Initial conditions.** The following takes the rules for a cyclic tag system in the form used on page 895 (with the restrictions in the note below), together with the initial conditions for the tag system, and yields a specification of initial conditions in rule 110 which will emulate it. This specification gives a list of three blocks $\{b_1, b_2, b_3\}$ and the final initial conditions consist of an infinite repetition of b_1 blocks, followed by b_2, followed by an infinite repetition of b_3 blocks. The b_1 blocks act like "clock pulses", b_2 encodes the initial conditions for the tag system and the b_3 blocks encode the rules for the tag system.

```
CTToR110[rules_ /;
  Select[rules, Mod[Length[#], 6] ≠ 0 &] == {}, init_] :=
 Module[{g1, g2, g3, nr = 0, x1, y1, sp}, g1 = Flatten[
   Map[If[# === {}, {{{2}}}, {{{1, 3, 5 - First[#]}}}, Table[
     {4, 5 - #[[n]]}, {n, 2, Length[#]}]]] &, rules] /. a_Integer :>
    Map[{d[#[[1]], #[[2]]], s[#[[3]]]} &, Partition[c[a], 3]], 4];
 g2 = g1 = MapThread[If[#1 === #2 === {d[22, 11], s3}, {d[
     20, 8], s3}, #1] &, {g1, RotateRight[g1, 6]}]; While[Mod[
   Apply[Plus, Map[#[[1, 2]] &, g2]], 30] ≠ 0, nr++; g2 = Join[
   g2, g1]]; y1 = g2[[1, 1, 2]] - 11; If[y1 < 0, y1 += 30]; Cases[
  Last[g2][[2]], s[d[x_, y1], _, _, a_] :> (x1 = x + Length[a])];
 g3 = Fold[sadd, {d[x1, y1], {}}, g2]; sp = Ceiling[5 Length[
     g3[[2]]]/(28 nr) + 2]; {Join[Fold[sadd, {d[17, 1], {}},
   Flatten[Table[{{d[sp 28 + 6, 1], s[5]}, {d[398, 1], s[5]},
     {d[342, 1], s[5]}, {d[370, 1], s[5]}}, {3}], 1]][[2]], bg[
   4, 11]], Flatten[Join[Table[bgi, {sp 2 + 1 + 24 Length[init]}],
   init /. {0 → init0, 1 → init1}, bg[1, 9], bg[6, 60 - g2[[1, 1, 1]] +
     g3[[1, 1]] + If[g2[[1, 1, 2]] < g3[[1, 2]], 8, 0]]]], g3[[2]]}]

s[1] = struct[{3, 0, 1, 10, 4, 8}, 2];

s[2] = struct[{3, 0, 1, 1, 619, 15}, 2];

s[3] = struct[{3, 0, 1, 10, 4956, 18}, 2];

s[4] = struct[{0, 0, 9, 10, 4, 8}];

s[5] = struct[{5, 0, 9, 14, 1, 1}];

{c[1], c[2]} = Map[Join[{22, 11, 3, 39, 3, 1}, #] &,
 {{63, 12, 2, 48, 5, 4, 29, 26, 4, 43, 26, 4, 23, 3, 4, 47, 4, 4},
  {87, 6, 2, 32, 2, 4, 13, 23, 4, 27, 16, 4}}];

{c[3], c[4], c[5]} = Map[Join[#, {4, 17, 22, 4,
    39, 27, 4, 47, 4, 4}] &, {{17, 22, 4, 23, 24, 4, 31, 29},
  {17, 22, 4, 47, 18, 4, 15, 19}, {41, 16, 4, 47, 18, 4, 15, 19}}]

{init0, init1} = Map[IntegerDigits[216 (# + 432 10^49), 2] &,
{24600556015465847173551005175056992262806506766 1,
 10437461654894668528970898304417565508898347096 45}]

bgi = IntegerDigits[9976, 2]

bg[s_, n_] := Array[bgi[[1 + Mod[# - 1, 14]]] &, n, s]

ev[s[d[x_, y_], pl_, pr_, b_]] := Module[{r, pl1, pr1}, r =
  Sign[BitAnd[2 ^ ListConvolve[{1, 2, 4}, Join[bg[pl - 2, 2], b,
     bg[pr, 2]]], 110]]; pl1 = (Position[r - bg[pl + 3, Length[r]],
     1 | -1] /. {} → {{Length[r]}})[[1, 1]]; pr1 = Max[pl1,
  (Position[r - bg[pr + 5 - Length[r], Length[r]], 1 | -1] /. {} →
     {{1}})[[-1, 1]]]; s[d[x + pl1 - 2, y + 1], pl1 + Mod[pl + 2, 14],
  1 + Mod[pr + 4, 14] + pr1 - Length[r], Take[r, {pl1, pr1}]]]

struct[{x_, y_, pl_, pr_, b_, bl_}, p_Integer : 1] := Module[
  {gr = s[d[x, y], pl, pr, IntegerDigits[b, 2, bl]], p2 = p + 1},
  Drop[NestWhile[Append[#, ev[Last[#]]] &, {gr},
    If[Rest[Last[#]] === Rest[gr], p2--]; p2 > 0 &], -1]]

sadd[{d[x_, y_], b_}, {d[dx_, dy_], st_}] :=
 Module[{x1 = dx - x, y1 = dy - y, b2, x2, y2}, While[y1 > 0,
   {x1, y1} += If[Length[st] == 30, {8, -30}, {-2, -3}]];
  b2 = First[Cases[st, s[d[x3_, -y1], pl_, _, sb_] :>
      Join[bg[pl - x1 - x3, x1 + x3], x2 = x3 + Length[sb];
       y2 = -y1; sb]]]; {d[x2, y2], Join[b, b2]}]
```

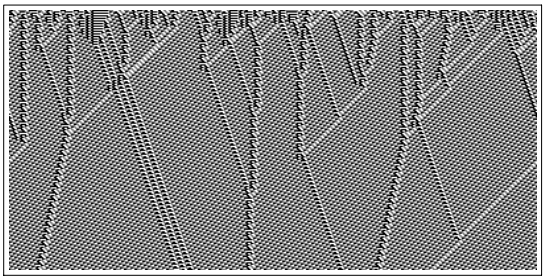
`CTToR110[{{}}, {1}]` yields blocks of lengths `{7204, 1873, 7088}`. But even `CTToR110[{{0, 0, 0, 0, 0, 0}, {}, {1, 1, 1, 1, 1, 1}, {}}, {1}]` already yields blocks of lengths `{105736, 34717, 95404}`. The picture below shows what happens if one chops these blocks into rows and arranges these in 2D arrays. In the first two blocks, much of what one sees is just padding to prevent clock pulses on the left from hitting data in the middle too early on any given step. The part of the middle block that actually encodes an initial condition grows like `180 Length[init]`. The core of the right-hand block grows approximately like `500 (Length[Flatten[rules]] + Length[rules])`, but to make a block that can just be repeated without shifts, between 1 and 30 repeats of this core can be needed.



■ **Page 689 · Tag systems.** The discussion in the main text and the construction above require a cyclic tag system with blocks that are a multiple of 6 long, and in which at least one block is added at some point in each complete cycle. By inserting `k = 6 Ceiling[Length[subs]/6]` in the definition of `TS1ToCT` from page 1113 one can construct a cyclic tag system of this kind to emulate any one-element-dependence tag system.

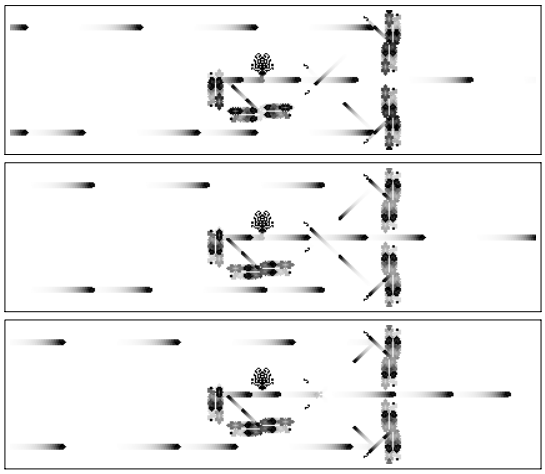
Class 4 Behavior and Universality

■ **2-neighbor rules.** Among 3-color 2-neighbor rules class 4 behavior seems to be comparatively rare; the picture at the top of the facing page shows an example with rule number 2144.



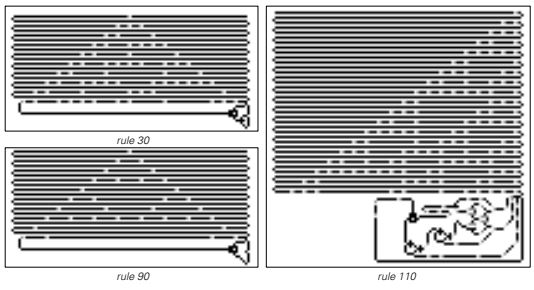
■ **Totalistic rules.** It is straightforward to show that totalistic cellular automata can be universal. Explicit simple candidates include $k = 2$, $r = 2$ rules with codes 20 and 52, as well as the various $k = 3$, $r = 1$ class 4 rules shown in Chapter 3.

■ **Page 693 · 2D cellular automata.** Universality was essentially built in explicitly to the underlying rules for the 2D cellular automaton constructed by John von Neumann in 1952 as a model for self-reproduction. For among the 29 possible states allowed for each cell were ones set up to behave quite directly like components for practical electronic computers like the EDVAC—as well as to grow new memory areas and so on. In the mid-1960s Edgar Codd showed that a system similar to von Neumann's could be constructed with only 8 possible states for each cell. Then in 1970 Roger Banks managed to show that the 2-state 5-neighbor symmetric 2D rule 4005091440 was able to reproduce all the same logical elements. (This system, like rule 110, requires an infinite repetitive background in order to support universality.) Following the invention of the Game of Life, considerable work was done in the early 1970s to identify structures that could be used to make the analog of logic circuits. John Conway worked on an explicit proof of universality based on emulating register machines, but this was apparently never completed. Yet by the 1980s it had come to be generally believed that the Game of Life had in fact been proved universal. No particularly rigorous treatments of the system were given, and the mere existence of configurations that can act for example like logic gates was often assumed immediately to imply universality. From the discoveries I have made, I have no doubt at all that the Game of Life is in the end universal, and indeed I believe that the kind of elaborate behavior needed to support various components is in fact good evidence for this. But the fact remains that a complete and rigorous proof of universality has apparently still never been given for the Game of Life. Particularly in recent years elaborate constructions have been made of for example Turing machines. But so far they have always had only a fixed number of elements on their tape, which is not sufficient for universality. Extending constructions is often very tricky; much as in rule 110 it is easy for there to be subtle bugs associated with rare mismatches in the placement of structures and timing of interactions. The pictures below nevertheless show a rather simple implementation of a NAND gate in Life. The input comes from the left encoded as the presence or absence of spaceships 92 cells apart. The spaceships are converted to gliders. When only one glider is present, a new spaceship emerges on the right as the output. But when two gliders are present, their collision forms a wall, which prevents output of the spaceship.



If one considers rules with more than two colors, it becomes straightforward to emulate standard logic circuits. The pictures below show how 1D cellular automata can be implemented in the 4-color WireWorld cellular automaton of Brian Silverman from 1987, whose rules find the new value of a cell from its old value a and the number u of its 8 neighbors that are 1's according to

$a /. \{0 \to 0,\ 1 \to 2,\ 2 \to 3,\ 3 :\to If[0 < u < 3,\ 1,\ 3]\}$



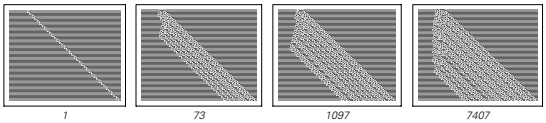
The Threshold of Universality in Cellular Automata

■ **Claims of non-universality.** Over the years, there have been a few erroneous claims of proofs that universality is impossible in particular kinds of simple cellular automata. The basic mistake is usually to make the implicit assumption that computation must be done in some rather specific way—that does not happen to be consistent with the way we have for example seen that it can be done in rule 110.

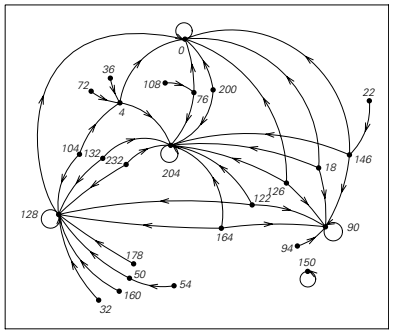
■ **Page 700 · Rule 73.** ■■■ on a white background yields a pattern that contains the last structure shown here.

■ **Page 700 · Rule 30.** For the first background shown, no initial region up to size 25 yields a truly localized structure, though for example □□□■□■□■■■ starts off growing quite slowly.

■ **Rule 41.** Various rules like rule 41 below can perhaps be viewed as having localized structures—though ones that apparently always travel in the same direction at the same speed. None of the first million initial conditions for rule 41 yield unbounded growth, though some can still generate fairly wide patterns, as in the pictures below. (The initial condition consisting □□■■■□□■■■■ repeated, followed by ■, followed by ■□□□□□ repeated nevertheless yields a region that grows forever.)



■ **Page 702 · Rule emulations.** The network below shows which quiescent symmetric elementary rules can emulate which with blocks of length 8 or less. (Compare page 269.)



In all cases things are set up so that several steps in one rule emulate a single step in another. The examples shown in detail in the main text all have the feature that the block size b and number of steps t are matched, so that $r\,t = b$ (where the range $r = 1$ for elementary rules). It is also possible to set up emulations where this equality does not hold—and indeed some of the cases listed in the main text and shown in the picture above are of this type. In those where $r\,t < b$ there are more cells that are in principle determined by a given set of initial blocks—but the outermost of these cells are ignored when the outcome for a particular cell is deduced. In cases where $r\,t > b$ there are more initial cells whose values are specified—but the outermost of these turn out to be irrelevant in determining the outcome for a particular cell. This lack of dependence makes it somewhat inevitable that the only rules that end up being emulated in this way are ones with very simple behavior.

In any 1D cellular automaton the color of a particular cell can always be determined from the colors t steps back of a block of $2\,r\,t + 1$ cells (compare pages 605 and 960). But such a block corresponds in a sense to a horizontal slice through the cone of previous cell colors. And it turns out also to be possible to determine the color of a particular cell from slices at essentially any rational angle corresponding to a propagation speed less than r. So this means that one can consider encodings based on blocks that have a kind of staircase shape—as in the rule 45 example shown.

■ **Encodings.** Generalizing the setup in the main text one can say that a cellular automaton i can emulate j if there is some encoding function ϕ that encodes the initial conditions a_j for j as initial conditions for i, and which has an inverse that decodes the outcome for i to give the outcome for j. With evolution functions f_i and f_j the requirement for the emulation to work is

$$f_j[a_j] == \text{InverseFunction}[\phi][f_i[\phi[a_j]]]$$

In the main text the encoding function is taken to have the form *Flatten[a /. rules]*—where *rules* are say *{1 → {1, 1}, 0 → {0, 0}}*—with the result that the decoding function for emulations that work is *Partition[ã, b] /. Map[Reverse, rules]*.

An immediate generalization is to allow *rules* to have a form like *{1 → {1, 1}, 1 → {1, 0}, 0 → {0, 0}}* in which several blocks are in effect allowed to serve as possible encodings for a single cell value. Another generalization is to allow blocks at a variety of angles (see above). In most cases, however, introducing these kinds of slightly more complicated encodings does not fundamentally seem to expand the set of rules that a given rule can emulate. But often it does allow the emulations to work with smaller blocks. And so, for example, with the setup shown in the main text, rule 54 can emulate rule 0 only with blocks of length $b = 6$. But if either multiple blocks or $\delta = 1$ are allowed, b can be reduced to 4, with *rules* being *{1 → {1, 1, 1, 1}, 0 → {0, 0, 0, 0}, 0 → {0, 1, 1, 1}}* and *{0 → {0, 1, 0, 0}, 1 → {0, 0, 1, 0}}* in the two cases.

Various questions about encoding functions ϕ have been studied over the past several decades in coding theory. The block-based encodings discussed so far here correspond to block codes. Convolutional codes (related to sequential cellular automata) are the other major class of codes studied in coding theory, but in their usual form these do not seem especially useful for our present purposes.

In the most general case the encoding function can involve an arbitrary terminating computation (see page 1126). But types of encoding functions that are at least somewhat powerful yet can realistically be sampled systematically may perhaps include those based on neighbor-dependent substitution systems, and on formal languages (finite automata and generalizations).

▪ **Logic operations and universality.** Knowing that the circuits in practical computers use only a small set of basic logic operations—often just *Nand*—it is sometimes assumed that if a particular system could be shown to emulate logic operations like *Nand*, then this would immediately establish its universality. But at least on the face of it, this is not correct. For somehow there also has to be a way to store arbitrarily large amounts of data—and to apply suitable combinations of *Nand* operations to it. Yet while practical computers have elaborate circuits containing huge numbers of *Nand* operations, we now know that for example simple cellular automata that can be implemented with just a few *Nand* operations (see page 619) are enough. And from what I have discovered in this book, it may well be that in fact most systems capable of supporting even a single *Nand* operation will actually turn out to be universal. But the point is that in any particular case this will not normally be an easy matter to demonstrate. (Compare page 807.)

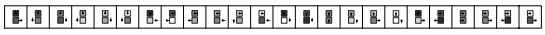
Universality in Turing Machines and Other Systems

▪ **Page 706 · Minsky's Turing machine.** The universal Turing machine shown was constructed by Marvin Minsky in 1962. If the rules for a one-element-dependence tag system are given in the form *{2, {{0, 1}, {0, 1, 1}}}* (compare page 1114), the initial conditions for the Turing machine are

```
TagToMTM[{2, rule_}, init_] :=
 With[{b = FoldList[Plus, 1, Map[Length, rule] + 1]},
  Drop[Flatten[{Reverse[Flatten[{1, Map[{Map[
          {1, 0, Table[0, {b[[# + 1]]}] &, #], 1} &, rule], 1}]],
    0, 0, Map[{Table[2, {b[[# + 1]]}], 3} &, init]}], -1]]
```

surrounded by *0*'s, with the head on the leftmost *2*, in state *1*. An element *-1* in the tag system corresponds to halting of the Turing machine. The different cases in the rules for the tag system are laid out on the left in the Turing machine. Each step of tag system evolution is implemented by having the head of the Turing machine scan as far to the left as it needs to get to the case of the tag system rule that applies—then copy the appropriate elements to the end of the sequence on the right. Note that although the Turing machine can emulate any number of colors in the tag system, it can only emulate directly rules that delete exactly 2 elements at each step. But since we know that at least with sufficiently many colors such tag systems are universal, it follows that the Turing machine is also universal.

▪ **History.** Alan Turing gave the first construction for a universal Turing machine in 1936. His construction was complicated and had several bugs. Claude Shannon showed in 1956 that 2 colors were sufficient so long as enough states were used. (See page 669; conversion of Minsky's machine using this method yields a *{43, 2}* machine.) After Minsky's 1962 result, comparatively little more was published about small universal Turing machines. In the 1980s and 1990s, however, Yurii Rogozhin found examples of universal Turing machines for which the number of states and number of colors were: *{24, 2}*, *{10, 3}*, *{7, 4}*, *{5, 5}*, *{4, 6}*, *{3, 10}*, and *{2, 18}*. The smallest product of these numbers is 24 (compare note below), and the rule he gave in this case is:



Note that these results concern Turing machines which can halt (see page 1137); the Turing machines that I consider do not typically have this feature.

▪ **Page 707 · Rule 110 Turing machines.** Given an initial condition for rule 110, the initial condition for the Turing machine shown here is obtained as *Prepend[4 list, 0]* with *1*'s on the left and *0*'s on the right. The Turing machine

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

with $s = 4$ states and $k = 3$ possible colors also emulates rule 110 when started from *Prepend[list + 1, 1]* surrounded by *0*'s. The $s = 3$, $k = 4$ Turing machine

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

started from *Append[list, 0]* with *0*'s on the left and *2*'s on the right generates a shifted version of rule 110. Note that this Turing machine requires only 8 out of the 12 possible cases in its rules to be specified.

▪ **Rule 60 Turing machines.** One can emulate rule 60 using the 8-case $s = 3$, $k = 3$ Turing machine (with initial condition *Append[list + 1, 1]* surrounded by *0*'s)

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

or by using the 6-case $s = 2$, $k = 4$ Turing machine (with initial condition *Append[3 list, 0]* with *0*'s on the left and *1*'s on the right)

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

This second Turing machine is directly analogous to the one for rule 110 on page 707. Random searches suggest that among $s = 3$, $k = 3$ Turing machines roughly one in 25 million reproduce rule 60 in the same way as the machines discussed here. (See also page 665.)

■ **Turing machine enumeration.** Of the 4096 $s = 2$, $k = 2$ Turing machines (see page 888) 560 are distinct after taking account of obvious symmetries and equivalences. Ignoring machines which cannot escape from one of their possible states or which yield motion in only one direction or cells of only one color leaves a total of 237 cases. If one now ignores machines that do not allow the head to move more than one step in one of the two directions, that always yield the same color when moving in a particular direction, or that always leave the tape unchanged, one is finally left with just 25 distinct cases.

Of the 2,985,984 $s = 3$, $k = 2$ machines, 125,294 survive after taking account of obvious symmetries and equivalences, while imposing analogs of the other conditions above yields in the end 16,400 distinct cases. For $s = 2$, $k = 3$ machines, the first two numbers are the same, but the final number of distinct cases is 48,505.

■ **States versus colors.** The total number of possible Turing machines depends on the product $s\,k$. The number of distinct machines that need to be considered increases as k increases for given $s\,k$ (see note above). $s = 1$ or $k = 1$ always yield trivial behavior. The fraction of machines that show non-repetitive behavior seems to increase roughly like $(s-1)(k-1)$ (see page 888). More complex behavior—and presumably also universality—seems however to occur slightly more often with larger k than with larger s.

■ ***s*=2, *k*=2 Turing machines.** As illustrated on page 761, even extremely simple Turing machines can have behavior that depends in a somewhat complicated way on initial conditions. Thus, for example, with the rule

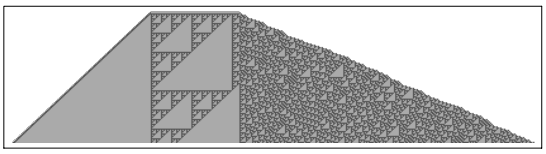
```
{{1, 0} → {1, 1, -1}, {1, 1} → {2, 1, 1},
 {2, 0} → {1, 0, -1}, {2, 1} → {1, 0, 1}}
```

the head moves to the right whenever the initial condition consists of odd-length blocks of 1's separated by single 0's; otherwise it stays in a fixed region.

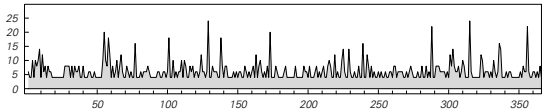
■ **Page 709 · Machine 596440.** For any list of initial colors *init*, it turns out that successive rows in the first t steps of the compressed evolution pattern turn out to be given by

```
NestList[Join[{0}, Mod[1 + Rest[FoldList[Plus, 0, #]], 2],
    {{0}, {1, 1, 0}}[[Mod[Apply[Plus, #], 2] + 1]]] &, init, t]
```

Inside the right-hand part of this pattern the cell values can then be obtained from an upside-down version of the rule 60 additive cellular automaton, and starting from a sequence of *1*'s the picture below shows that a typical rule 60 nested pattern can be produced, at least in a limited region.



The presence of glitches on the right-hand edge of the whole pattern means, however, that overall there is nothing as simple as nested behavior—making it conceivable that (possibly with analogies to tag systems) behavior complex enough to support universality can occur. The plot below shows the distances between successive outward glitches on the right-hand side; considerable complexity is evident.



■ **Page 710 · *s*=3, *k*=2 Turing machines.** Compare page 763 and particularly the discussion of machine 600720 on page 1145.

■ **Tag systems.** Marvin Minsky showed in 1961 that one-element-dependence tag systems (see page 670) can be universal. Hao Wang in 1963 constructed an example that deletes just 2 elements at each step and adds at most 3 elements—but has a large number of colors. I suspect that universal examples with blocks of the same size exist with just 3 colors.

■ **Encoding sequences by integers.** In many constructions it is useful to be able to encode a list of integers of any length by a single integer. (See e.g. page 1127.) One way to do this is by using the Gödel number *Product[Prime[i]^list[[i]], {i, Length[list]}]*. An alternative is to use the Chinese Remainder Theorem. Given *p = Array[Prime, Length[list], PrimePi[Max[list]] + 1]* or any list of integers that are all relatively prime and above *Max[list]* (the integers in *list* are assumed positive)

```
CRT[list_, p_] :=
  With[{m = Apply[Times, p]}, Mod[Apply[Plus,
      MapThread[#1 (m/#2)^EulerPhi[#2] &, {list, p}]], m]]
```

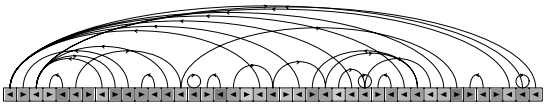
yields a number x such that *Mod[x, p] == list*. Based on this

```
LE[list_] := Module[{n = Length[list], i = Max[MapIndexed[
    #1 - #2 &, PrimePi[list]]] + 1}, CRT[PadRight[
  list, n + i], Join[Array[Prime[i + #] &, n], Array[Prime, i]]]]
```

will yield a number x that can be decoded into a list of length n using essentially the so-called Gödel β function

```
Mod[x, Prime[Rest[NestList[NestWhile[# + 1 &,
        # + 1, Mod[x, Prime[#]] == 0 &] &, 0, n]]]]
```

■ **Register machines.** The results of page 100 suggest that with 2 registers and up to 8 instructions no universal register machines (URMs) exist. Using the method of page 672 one can construct a URM with 3 registers and 175 instructions (or 2 registers and 4694 instructions) that emulates the universal Turing machine on page 706. Using work by Ivan Korec from the 1980s and 1990s one can also construct URMs which directly emulate other register machines. An example with 8 registers and 41 instructions is:



or

```
{d[4, 40], i[5], d[3, 9], i[3], d[7, 4], d[5, 14], i[6],
  d[3, 3], i[7], d[6, 2], i[6], d[5, 11], d[6, 3], d[4, 35],
  d[6, 15], i[4], d[8, 16], d[5, 21], i[1], d[3, 1], d[5, 25],
  i[2], d[3, 1], i[6], d[5, 32], d[1, 28], d[3, 1], d[4, 28],
  i[4], d[6, 29], d[3, 1], d[5, 24], d[2, 28], d[3, 1],
  i[8], i[6], d[5, 36], i[6], d[3, 3], d[6, 40], d[4, 3]}
```

Given any register machine, one first applies the function *RMToRM2* from page 1114, then takes the resulting program and initial condition and finds an initial condition for the URM using

```
R2ToURM[prog_, init_] := Join[init, With[
   {n = Length[prog]}, {1 + LE[Reverse[prog] /. {i[x_] → x,
       d[x_, y_] → 4 + 2 n + x - 2 y}], n + 1, 0, 0, 0, 0}]]
```

For the first example on page 98 this gives *{0, 0, 211680, 3, 0, 0, 0, 0}*. The process of emulation is quite slow, with each emulated step in this example taking about 20 million URM steps.

■ **Recursive functions.** The general recursive functions from page 907 provided an early example of universality (see page 907). That such functions are universal can be demonstrated by showing for example that they can emulate any tag system. With the state of a 2-color tag system encoded as an integer according to *FromDigits[Reverse[list] + 1, 3]* the following takes the rule for any such tag system (in the first form from page 894) and yields a primitive recursive function that emulates a single step in its evolution:

```
TSToPR[{n_, rule_}] := Fold[Apply[c, Flatten[{#1, Array[p, #
  2], c[r[z, c[r[p[1], s], c[r[z, p[2]], c[r[z, r[c[s, z], c[r[c[s,
  c[s, z]], z], p[2]]]], p[2]]], p[1]]], p[#2]]}]] &, c[c[r[p[1],
  s], p[1], c[r[p[1], r[z, c[s, c[s, s]]]], c[c[r[z, c[r[p[1], s],
  c[r[z, c[s, z]], c[r[p[1], r[z, c[r[p[1], s], c[r[z, p[2]], c[
   r[z, r[c[s, z], c[r[c[s, c[s, z]], z], p[2]]]], p[2]]], p[1]]]],
  p[2], p[3]]], p[1]]], p[1], p[1]], p[1]], p[2]]], p[n + 1],
  MapIndexed[c[r[z, c[r[p[1], p[4]], p[2], p[3], p[4]]], c[r[z,
   r[c[s, z], c[r[c[s, c[s, z]], z], p[2]]]], p[Length[#2] + 1]], #
   1[[1]], #1[[2]]] &, Nest[Partition[#, 2] &, Table[Nest[c[s, #] &
   z, FromDigits[Reverse[IntegerDigits[i, 2, n] /. rule] + 1, 3]],
   {i, 0, 2^n - 1}], n - 1], {0, n - 1}]], Range[n, 1, -1]]
```

(For tag system (a) from page 94 this yields a primitive recursive function of size 325.) The result of t steps of evolution is in general given in terms of this function f by *Nest[f, init, t]*, or equivalently *r[p[1], f][t, init]*. Any fixed number of steps of evolution can thus be emulated by applying a primitive recursive function. But if one wants to find out what happens after an arbitrarily large number of steps, one needs to use the μ operator, yielding a general recursive function. (So for example *μ[r[p[1], f]][init]* returns the smallest t for which the tag system reaches state *{}*—and never returns if the tag system does not halt.) Note that the same basic approach can be used to emulate Turing machines with recursive functions; the Turing machine configuration *{s, list, n}* can be encoded by a integer such as

```
2^FromDigits[Reverse[Take[list, n - 1]]]
  3^FromDigits[Take[list, {n + 1, -1}]] 5^list[[n]] 7^s
```

■ **Lambda calculus.** Formulations of recursive function theory from the 1920s and before tended to be based on making explicit definitions like those in the note above. But in the so-called lambda calculus of Alonzo Church from around 1930 what were instead used were pure functions such as *s = Function[x, x + 1]* and *plus = Function[{x, y}, If[x == 0, y, s[plus[x - 1, y]]]]*—of just the kind now familiar from *Mathematica*. Note that the explicit names of ("bound") variables in such pure functions are never significant—which is why in *Mathematica* one can for example use *s = # + 1 &*. (See page 907.)

The definitions in the note above involve both symbolic functions and literal integers. In the so-called pure lambda calculus integers are represented by symbolic expressions. The typical way this is done is to say that a function f_n corresponds to an integer n if $f_n[a][b]$ yields *Nest[a, b, n]* (see note below).

■ **Page 711 · Combinators.** After it became widely known in the 1910s that *Nand* could be used to build up any expression in basic logic (see page 1173) Moses Schönfinkel introduced combinators in 1920 with the idea of providing an analogous way to build up functions—and to remove any mention of variables—particularly in predicate logic (see page 898). Given the combinator rules

```
crules = {s[x_][y_][z_] → x[z][y[z]], k[x_][y_] → x}
```

the setup was that any function f would be written as some combination of s and k—which Schönfinkel referred to respectively as "fusion" and "constancy"—and then the result of applying the function to an argument x would be

given by f[x] //. crules. (Multiple arguments were handled for example as f[x][y][z] in what became known as "currying".) A very simple example of a combinator is id = s[k][k], which corresponds to the identity function, since id[x] //. crules yields x for any x. (In general any combinator of the form s[k][_] will also work.) Another example of a combinator is b = s[k[s]][k], for which b[x][y][z] //. crules yields x[y[z]].

With the development of lambda calculus in the early 1930s it became clear that given any expression expr such as x[y[x][z]] with a list of variables vars such as {x, y, z} one can always find a combinator equivalent to a lambda function such as Function[x, Function[y, Function[z, x[y[x][z]]]]], and it turns out that this can be done simply using

```
ToC[expr_, vars_] := Fold[rm, expr, Reverse[vars]]
rm[v_, v_] = id
rm[f_[v_], v_] /; FreeQ[f, v] = f
rm[h_, v_] /; FreeQ[h, v] = k[h]
rm[f_[g_], v_] := s[rm[f, v]][rm[g, v]]
```

So this shows that any lambda function can in effect be written in terms of combinators, without anything analogous to variables ever explicitly having to be introduced. And based on the result that lambda functions can represent recursive functions, which can in turn represent Turing machines (see note above), it has been known since the mid-1930s that combinators are universal. The rule 110 combinator on page 713 provides however a much more direct proof of this.

The usual approach to working with combinators involves building up arithmetic constructs from them. This typically begins by using so-called Church numerals (based on work by Alonzo Church on lambda calculus), and defining a combinator e_n to correspond to an integer n if e_n[a][b] //. crules yields Nest[a, b, n]. (The e on page 103 can thus be considered a Church numeral for 2 since e[a][b] is a[a[b]].) This can be achieved by taking e_n to be Nest[inc, zero, n] where

```
zero = s[k]
inc = s[s[k[s]][k]]
```

With this setup one then finds

```
plus = s[k[s]][s[k[s[k[s]]]][s[k[k]]]]
times = s[k[s]][k]
power = s[k[s[s[k][k]]]][k]
```

(Note that power[x][y] //. crules is y[x], and that by analogy x[x[y]] corresponds to y^{x^2}, x[y[x]] to $x^{x y}$, x[y][x] to x^{y^x}, and so on.)

Another approach involves representing integers directly as combinator expressions. As an example, one can take n to be represented just by Nest[s, k, n]. And one can then convert any Church numeral x to this representation by applying s[s[s[k][k]][k[s]]][k[k]]. To go the other way, one uses the result that for all Church numerals x and y, Nest[s, k, n][x][y] is also a Church numeral—as can be seen recursively by noting its equality to Nest[s, k, n - 1][y][x[y]], where as above x[y] is power[y][x]. And from this it follows that Nest[s, k, n] can be converted to the Church numeral for n by applying

```
s[s[s[s[s[k][k]][k[s[s[k[s]][k]][s[k][k]]]]][
k[s[s[k[s]][k]][s[s[k[s]][k]][s[k][k]]]]]][s[s[k[s]][
s[s[k[s]][s[k[s[s[s[s[s[s[k][k]][k[s]]][k[k]]][k[s[s[
k[s]][k]][s[k][k]]]]][k[s[s[k[s]][k]][s[s[k[s]][k]][s[k][
k]]]]]][k[s[s[s[s[k][k]][k[s[s[k[s]][s[k[s[s[k][k]]]][s[
k[k]][s[k[s[s[k[s]][k]]]][s[s[k][k]][k[k]]]]]]][s[k[k]][s[
s[k][k]][k[k]]]]]]][k[s[s[s[k][k]][k[s[k]]]][k[s[k]]]]]][
k[s[k]]]]]]]][s[k[k]][s[s[s[k][k]][k[s[s[k[s]][k]][s[k][
k]]]]][k[s[s[k[s]][k]][s[s[k[s]][k]][s[k][k]]]]]]]]][
k[s[k[k]][s[s[k[s]][k]]]]]]][k[s[k][k]]]]][k[s[k]]]
```

Using this one can find from the corresponding results for Church numerals combinator expressions for plus, times and power—with sizes 377, 378 and 382 respectively. It seems certain that vastly simpler combinator expressions will also work, but searches indicate that if inc has size less than 4, plus must have size at least 8. (Searches based on other representations for integers have also not yielded much. With n represented by Nest[k, s[k][k], n], however, s[s[s[s]][k]][k] serves as a decrement function, and with n represented by Nest[s[s], s[k], n], s[s[s][k]][k[k[s[s]]]] serves as a doubling function.

■ **Page 712 · Combinator properties.** The size of a combinator expression is conveniently measured by its LeafCount. If the evolution of a combinator expression reaches a fixed point, then the expression generated is always the same (Church-Rosser property). But the behavior in the course of the evolution can depend on how the combinator rules are applied; here expr /. crules is used at each step, as in the symbolic systems of page 896. The total number of combinator expressions of successively greater sizes is {2, 4, 16, 80, 448, 2688, 16896, 109824, …} (or in general 2^n Binomial[2 n - 2, n - 1]/n; see page 897). Of these, {2, 4, 12, 40, 144, 544, 2128, 8544, …} are themselves fixed points. Of combinator expressions up to size 6 all evolve to fixed points, in at most {1, 1, 2, 3, 4, 7} steps respectively (compare case (a)); the largest fixed points have sizes {1, 2, 3, 4, 6, 10} (compare case (b)). At size 7, all but 2 of the 16,896 possible combinator expressions evolve to fixed points, in at most 12 steps (case (c)). The largest fixed point has size 41 (case (d)). s[s[s]][s][s][s][s] (case (e)) and s[s][s][s[s]][s][s] lead to expressions that grow like $2^{t/2}$. The maximum number of levels in these expressions (see

page 897) grows roughly linearly, although *Depth[expr]* reaches 14 after 26 and 25 steps, then stays there. At size 8, out of all 109,824 combinator expressions it appears that 49 show exponential growth, and many more show roughly linear growth. *s[s][k][s[s[s]][s]][s]* goes to a fixed point of size 80. *s[s[s]][s][s][s][s[k]]* (case (i)) increases rapidly to size 7050 but then repeats with period 3. *s[s[s[s][s]]][s][s][k]* (case (j)) grows to a maximum size of 1263, but then after 98 steps evolves to a fixed point of size 17. For *s[s][k][s[s[s][k]]][k]* (case (k)) the size at step $t - 7$ is given by

```
h[1] = h[2] = h[3] = 12

h[t_] := If[Mod[t, 4] == 2, 2, 1] (h[Ceiling[t/2] - 1] + t) +
    {3, 5, -7, -1}[[Mod[t, 4] + 1]]
```

Examples with similar behavior are *s[s[s][k]][s][s[s][k]]*, *s[s[s]][s][s[s][k]][k]* and *s[s[s][s]][s][s[s][k]]*. Among those with roughly exponential growth but seemingly random fluctuations are *s[s[s[s]]][s][s][s][k]*, *s[s[s]][s][s[s][s]][k]* and *s[s[s[s]][s[s]][k][s]*.

■ **Single combinators.** As already noted by Moses Schönfinkel in 1920, it is possible to set up combinator systems with just a single combinator. In such cases, combinator expressions can be viewed as binary trees without labels, equivalent to balanced strings of parentheses (see page 989) or sequences of 0's and 1's. One example of a single combinator system can be found using *{s → j[j], k → j[j[j]]}*, and has combinator rules (whose order matters):

```
{j[j][x_][y_][z_] → x[z][y[z]], j[j[j]][x_][y_] → x}
```

The smallest initial conditions in this case that lead to unbounded growth are of size 14; two are versions of those for *s*, *k* combinators above, while the third is *j[j][j[j]][j[j]][j[j][j[j]]]][j[j][j]]*.

The forms *j[j]* and *j[j[j]]* appear to be the simplest that can be used for *s* and *k*; *j* and *j[j]*, for example, do not work.

■ **Page 714 · Cellular automaton combinators.** With *k* and *s[k]* representing respectively cell values *0* and *1*, a combinator *f* for which $f[a_{-1}][a_0][a_1]$ gives the new value of a single cell in an elementary cellular automaton with rule number *m* can be constructed as

```
Apply[p[p[p[#1][#2]][p[#3][#4]]][p[p[#5][#6]][p[#7][
 #8]]] /. {0 → k, 1 → s[k]} &, IntegerDigits[m, 2, 8]] //. crules
```

where

```
p = ToC[z[y][x], {x, y, z}] //. crules
```

The resulting combinator has size 61, but for specific rules somewhat smaller combinators can be found—an example for rule 90 is *s[k[k]][s[s][k[s[s[s[k][k]][k[s[k]]]][k[k]]]]]* with size 16.

To emulate cellular automaton evolution one starts by encoding a list of cell values by the single combinator

```
p[num[Length[list]]][
    Fold[p[Nest[s, k, #2]][#1] &, p[k][k], list]] //. crules
```

where

```
num[n_] := Nest[inc, s[k], n]

inc = s[s[k[s]][k]]
```

One can recover the original list by using

```
Extract[expr, Map[Reverse[IntegerDigits[#, 2]] &,
  3 + 59/15 (16 ^ Range[Depth[expr[s[k]][s][k] //. crules] -
      1, 1, -1] - 1)]] /. {k → 0, s[k] → 1}
```

In terms of the combinator *f* a single complete step of cellular automaton evolution can be represented by

```
w = cr[p[inc[inc[x[s[k]]]]][
 inc[x[s[k]]][cr[p[y[s[k]][k]][p[y[s[k]][s[k]]][y[k]]],
  {y}]][p[x[s[k]][cr[p[p[f[y[k][k][k][s[k]]]][
     y[k][k][s[k]]]][y[k][s[k]]]][y[s[k]]]][y[k][k]], {y}]][
  p[p[k][k]][p[k][x[k]]]][s[k]]][p[k][p[k][k]]]][k]], {x}]

cr[expr_, vars_] := ToC[expr //. crules, vars]
```

where there is padding with *0* on either side. With this setup *t* steps of evolution are given simply by *Nest[w, init, t]*. With an initial condition of *n* cells, this then takes roughly $(100 + 35\,n)\,t + 33\,t^2$ steps of combinator evolution.

■ **Testing universality.** One can tell that a symbolic system is universal if one can find expressions that act like the *s* and *k* combinators, so that, for example, for some expression *e*, *e[x][y][z]* evolves to *x[z][y[z]]*.

■ **Criteria for universality.** See page 1126.

■ **Classes of systems.** This chapter has shown that various individual systems with fixed rules exhibit universality when suitable initial conditions are chosen. One can also consider whole classes of systems in which rules as well as initial conditions can be chosen. And then one can say for example that as a class of systems cellular automata are universal, but neighbor-independent substitution systems are not.

The Principle of Computational Equivalence

Basic Framework

▪ **All is computation.** The early history of science includes many examples of attempts to treat all aspects of the universe in a uniform way. Some were more successful than others. "All is fire" was never definite enough to lead to much, but "all is number" can be viewed as an antecedent to the whole application of mathematics to science, and "all is atoms" to the atomic theory of matter and quantum mechanics. My "all is computation" will, I believe, form the basis for a fruitful new direction in science. It should be pointed out, however, that it is wrong to think that once one has described everything as, say, computation, then there is nothing more to do. Indeed, the phenomenon of computational irreducibility discussed in this chapter specifically implies that in many cases irreducible work has to be done in order to find out how any particular system will behave.

Outline of the Principle

▪ **Note for mathematicians.** The way I discuss the Principle of Computational Equivalence is in a sense opposite to what would be typical in modern mathematics. For rather than starting with very specific definitions and then expanding from these, I start from general intuition and then use this to come up with more specific results. In the years to come there will no doubt be many attempts to formulate parts of the Principle of Computational Equivalence in ways that are closer to the traditions of modern mathematics. But at least at first, I suspect that huge simplifications will be made, with the result that all sorts of misleading conclusions will probably be reached, perhaps in some cases even seemingly contradicting the principle.

▪ **History.** As I discuss elsewhere, aspects of the Principle of Computational Equivalence have many antecedents. But the complete principle is presented for the first time in this book, and is the result of thinking I did in the late 1980s and early 1990s.

▪ **Page 717 · Church's Thesis.** The idea that any computation that can be done at all can be done by a universal system such as a universal Turing machine is often referred to as Church's Thesis. Following the introduction of so-called primitive recursive functions (see page 907) in the 1880s, there had by the 1920s emerged the idea that perhaps any reasonable function could be computed using the small set of operations on which primitive recursive functions are based. This notion was supported by the fact that certain modifications to these operations were found to allow only the exact same set of functions. But the discovery of the Ackermann function in the late 1920s (see page 906) showed that there are reasonable functions that are not primitive recursive. The proof of Gödel's Theorem in 1931 made use of so-called general recursive functions (see page 1121) as a way to represent possible functions in arithmetic. And in the early 1930s the two basic idealizations used in foundational studies of mathematical processes were then general recursive functions and lambda calculus (see page 1121). By 1934 these were known to be equivalent, and in 1935 Alonzo Church suggested that either of them could be used to do any mathematical calculation which could effectively be done. (It had been noted that many specific kinds of calculations could be done within such systems—and that processes like diagonalization led to operations of a seemingly rather different character.) In 1936 Alan Turing then introduced the idea of Turing machines, and argued that any mathematical process that could be carried out in practice, say by a person, could be carried out by a Turing machine. Turing proved that his machines were exactly equivalent in their computational capabilities to lambda calculus. By the 1940s Emil Post had shown that the string rewriting systems he had studied were also equivalent, and as electronic computers began to be developed it became quite firmly established that Turing machines provided an appropriate idealization for what computations could be done. From the 1940s to 1960s many different types of systems—almost all mentioned at some

point or another in this book—were shown to be equivalent in their computational capabilities. (Starting in the 1970s, as discussed on page 1143, emphasis shifted to studies not of overall equivalence but instead equivalence with respect to classes of transformations such as polynomial time.)

When textbooks of computer science began to be written some confusion developed about the character of Church's Thesis: was it something that could somehow be deduced, or was it instead essentially just a definition of computability? Turing and Post seem to have thought of Church's Thesis as characterizing the "mathematicizing power" of humans, and Turing at least seems to have thought that it might not apply to continuous processes in physics. Kurt Gödel privately discussed the question of whether the universe could be viewed as following Church's Thesis and being "mechanical". And starting in the 1950s a few physicists, notably Richard Feynman, asked about fundamental comparisons between computational and physical processes. But it was not until the 1980s—perhaps particularly following some of my work—that it began to be more widely realized that Church's Thesis should best be considered a statement about nature and about the kinds of computations that can be done in our universe. The validity of Church's Thesis has long been taken more or less for granted by computer scientists, but among physicists there are still nagging doubts, mostly revolving around the perfect continua assumed in space and quantum mechanics in the traditional formalism of theoretical physics (see page 730). Such doubts will in the end only be put to rest by the explicit construction of a discrete fundamental theory along the lines I discuss in Chapter 9.

The Content of the Principle

■ **Page 719 · Character of principles.** Examples of principles that can be viewed in several ways include the Principle of Entropy Increase (Second Law of Thermodynamics), the Principle of Relativity, Newton's Laws, the Uncertainty Principle and the Principle of Natural Selection. The Principle of Entropy Increase, for example, is partly a law of nature relating to properties of heat, partly an abstract fact about ensembles of dynamical systems, and partly a foundation for the definition of entropy. In this case and in others, however, the most important role of a principle is as a guide to intuition and understanding.

■ **Page 720 · Oracles.** Following his introduction of Turing machines Alan Turing tried in 1937 to develop models that would somehow allow the ultimate result of absolutely every conceivable computation to be determined. And as a step towards this, he introduced the idea of oracles which would give results of computations that could not be found by any Turing machine in any limited number of steps. He then noted, for example, that if an oracle were set up that could answer the question for a particular universal system of whether that system would ever halt when given any specific input, then with an appropriate transformation of input this same oracle could also answer the question for any other system that can be emulated by the universal system. But it turns out that this is no longer true if one allows systems which themselves can access the oracle in the course of their evolution. Yet one can then imagine a higher-level oracle for these systems, and indeed a whole hierarchy of levels of oracles—as studied in the theory of degrees of unsolvability. (Note that for example to answer the question of whether or not a given Turing machine always halts can require a second-order oracle, since it is a Π_2 question in the sense of page 1139.)

■ **Initial conditions.** Oracles are usually imagined as being included in the internal rules for a system. But if there are an infinite number of elements that can be specified in the initial condition—as in a cellular automaton—then a table for an oracle could also be given in the initial conditions.

■ **Page 722 · Criteria for universality.** To be universal a system must in effect be able to emulate any feature of any system. So at some level any feature can be thought of as a criterion for universality. Some features—like the possibility of information transmission—may be more obvious than others, but despite occasional assertions to the contrary in the scientific literature none is ever the whole story. Since any given universal system must be able to emulate any other universal system it follows that within any such system it must in a sense be possible to find any known universal system. But inevitably the encoding will sometimes be very complicated. And in practice if there are many simple rules that are universal they cannot all be related by simple encodings. (See also the end of Chapter 11.)

■ **Page 722 · Encodings.** One can prevent an encoding from itself introducing universality by insisting, for example, that it be primitive recursive (see page 907) or always involve only a bounded number of steps. One can also do this—as in the rule 110 proof in the previous chapter—by having programs and data be encoded separately, and appear, say, as distinct parts of the initial conditions for the system one is studying. (See also page 1118.)

■ **Density of universal systems.** One might imagine that it would be possible to make estimates of the overall density of universal systems, perhaps using arguments like those for the density of primes, or for the density of algorithmically

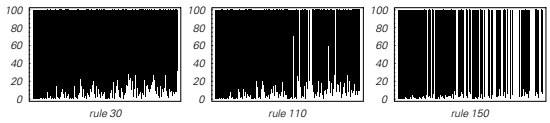
random sequences. But as it turns out I know of no way to make any such estimates. If one has shown that various simple rules are universal, then it follows that rules which generalize these must also be universal. But even from this I do not know, for example, how to prove that the density of universal rules cannot decrease when rules become more complicated.

■ **Page 723 · Proving universality.** The question of whether a system is universal is in general undecidable. Using a specific mathematical axiom system such as Peano arithmetic or set theory it may also be that there is no proof that can be given. (It is straightforward to construct complicated examples where this is the case.) In practice it seems to get more difficult to prove universality when the structure of a system gets simpler. Current proofs of universality all work by showing how to emulate a known universal system. Some level of checking can be done by tracing the emulation of random initial conditions for the universal system. In the future it seems likely that automated theorem-proving methods should help in finding proofs of universality.

■ **Page 724 · History.** There are various precedents in philosophy and mysticism for the idea of encoding all possible knowledge of some kind in a single object. An example in computation theory is the concept emphasized by Gregory Chaitin of a number whose n^{th} digit specifies whether a computation with initial condition n in a particular system will ever halt. This particular number is far from being computable (see page 1128), as a result of the undecidability of the halting problem (see page 754). But a finite version in which one looks at results after a limited number of steps is similar to my concept of a universal object. (See also page 1067.)

■ **Page 725 · Universal objects.** A more direct way to create a universal object is to set up, say, a 4D array in which two of the dimensions range respectively over possible 1D cellular automaton rules and over possible initial conditions, while the other two dimensions correspond to space and time in the evolution of each cellular automaton from each initial condition. (Compare the parameter space sets of page 1006.)

■ **Page 725 · Block occurrences.** The pictures below show at which step each successive block of length up to 8 first appears in evolution according to various cellular automaton rules starting from a single black cell. For rule 30, the numbers of steps needed for each block of lengths 1 through 10 to appear at least once is *{1, 2, 4, 12, 22, 24, 33, 59, 69, 113}*. (See also page 871.)



The Validity of the Principle

■ **Page 729 · Continuum and cardinality.** Some notion of a distinction between continuous and discrete systems has existed since antiquity. But in the 1870s the distinction became more precise with Georg Cantor's characterization of the total numbers of possible objects of various types in terms of different orders of infinity (see page 1162). The total number of possible integers corresponds to the smallest level of infinity, usually denoted $\aleph_0$. The total number of possible lists of integers of given finite length—and thus the number of possible rational numbers—turns out also to be $\aleph_0$. The reason is that it is always possible to encode any finite list of integers as a single integer, as discussed on page 1120. (A way to do this for pairs of non-negative integers is to use *σ[{x_, y_}] := 1/2 (x + y) (x + y + 1) + x*.) But for real numbers the story is different. Any real number x can be represented as a set of integers using for example

Rest[FoldList[Plus, 1, ContinuedFraction[x]]]

but except when x is rational this list is not finite. Since the number of possible subsets of a set with k elements is 2^k, the number of possible real numbers is $2^{\aleph_0}$. And using Cantor's diagonal argument (see note below) one can then show that this must be larger than $\aleph_0$. (The claim that there are no sets intermediate in size between $\aleph_0$ and $2^{\aleph_0}$ is the so-called continuum hypothesis, which is known to be independent of the standard axioms of set theory, as discussed on page 1155.) Much as for integers, finite lists of real numbers can be encoded as single real numbers—using for example roughly *FromDigits[Flatten[Transpose[RealDigits[list]]]]*—so that the number of such lists is $2^{\aleph_0}$. (Space-filling curves yield a more continuous version of such an encoding.) But unlike for integers the same turns out to be true even for infinite lists of real numbers. (The function σ above can for example be used to specify the order in which to sample elements in *RealDigits[list]*). The total number of possible functions of real numbers is $2^{2^{\aleph_0}}$; the number of continuous such functions (which can always be represented by a list of coefficients for a series) is however only $2^{\aleph_0}$.

In systems like cellular automata, finite arrangements of black cells on a background of white cells can readily be specified by single integers, so the number of them is $\aleph_0$. But infinite configurations of cells are like digit sequences of real

numbers (as discussed on page 869 they correspond more precisely to elements in a Cantor set), so the number of them is $2^{\aleph_0}$. Continuous cellular automata (see page 155) also have $2^{\aleph_0}$ possible states.

■ **Computable reals.** The stated purpose of Alan Turing's original 1936 paper on computation was to introduce the notion of computable real numbers, whose n^{th} digit for any n could be found by a Turing machine in a finite number of steps. Real numbers used in any explicit way in traditional mathematics are always computable in this sense. But as Turing pointed out, the overwhelming majority of all possible real numbers are not computable. For certainly there can be no more computable real numbers than there are possible Turing machines. But with his discovery of universality, Turing established that any Turing machine can be emulated by a single universal Turing machine with suitable initial conditions. And the point is that any such initial conditions can always be encoded as an integer.

As examples of non-computable reals that can readily be defined, Turing considered numbers whose successive digits are determined by the eventual behavior after an infinitely long time of a universal system with successive possible initial conditions (compare page 964). With two possible forms of behavior `h[i] = 0` or `1` for initial condition i, an example of such a number is `Sum[2^-i h[i], {i, ∞}]`. Closely related is the total probability for each form of behavior, given for example by `Sum[2^(-Ceiling[Log[2, i]]) h[i], {i, ∞}]`. I suspect that many limiting properties of systems like cellular automata in general correspond to non-computable reals. An example is the average density of black cells after an arbitrarily long time. For many rules, this converges rapidly to a definite value; but for some rules it will wiggle forever as more and more initial conditions are included in the average.

■ **Diagonal arguments.** Similar arguments were used by Georg Cantor in 1891 to show that there must be more real numbers than integers and by Alan Turing in 1936 to show that the problem of enumerating computable real numbers is unsolvable. One might imagine that it should be possible to set up a function `f[i, n]` which if given successive integers i would give the n^{th} base 2 digit in every possible real number. But what about the number whose n^{th} digit is `1 - f[n, n]`? This is still a real number, yet it cannot be generated by `f[i, n]` for any i—thus showing that there are more real numbers than integers. Analogously, one might imagine that it should be possible to have a function `f[i, n]` which enumerates all possible programs that always halt, and specifies a digit in their output when given input n. But what about the program with output `1 - f[n, n]`? This program always halts, yet it does not correspond to any possible value of i—even though universality implies that any program should be encodable by a single integer i. And the only possible conclusion from this is that `f[i, n]` cannot in fact be implemented as a program that always halts—thus demonstrating that the computable real numbers cannot explicitly be enumerated. (Closely related is the undecidability of the problem discussed on page 1137 of whether a system halts given any particular input.) (See also pages 907 and 1162.)

■ **Continuous computation.** Various models of computation that involve continuous elements have been proposed since the 1930s, and unlike those with discrete elements they have often not proved ultimately equivalent. One general class of models based on the work of Alan Turing in 1936 follow the operation of standard digital computers, and involve looking at real numbers in terms of digits, and using discrete processes to generate these digits. Such models inevitably handle only computable reals (in the sense defined above), and can never do computations beyond those possible in ordinary discrete systems. Functions are usually considered computable in such models if one can take the procedure for finding the digits of x and get a procedure for finding the digits of `f[x]`. And with this definition all standard mathematical functions are computable—even those from chaos theory that excavate digits rapidly. (It seems possible however to construct functions computable in this sense whose derivatives are not computable.) The same basic approach can be used whenever numbers are represented by constructs with discrete elements (see page 143), including for example symbolic formulas.

Several times since the 1940s it has been suggested that models of computation should be closer to traditional continuous mathematics, and should look at real numbers as a whole, not in terms of their digit or other representations. In a typical case, what is done is to generalize the register machines of page 97 to have registers that hold arbitrary real numbers. It is then usually assumed, however, that the primitive operations performed on these registers are just those of ordinary arithmetic, with the result that only a very limited set of functions (not including for example the exponential function) can be computed in a finite number of steps. Introducing other standard mathematical functions as primitives does not usually help much, unless one somehow gives the system the capability to solve any equation immediately (see below). (Other appropriate primitives may conceivably be related to the solubility of Hilbert's Thirteenth Problem and the fact that any continuous function with any number of arguments can be written as a one-argument function of a sum of a handful of fixed one-argument functions applied to the arguments of the original function.)

Most of the types of programs that I have discussed in this book can be generalized to allow continuous data, often just by having a continuous range of values for their elements (see e.g. page 155). But the programs themselves normally remain discrete, typically involving discrete choices made at discrete steps. If one has a table of choices, one can imagine generalizing this to a function of a real number. But to specify this function one normally has no choice but to use some type of finite formula. And to set up any kind of continuous evolution, the most obvious approach is to use traditional mathematical ideas of calculus and differential equations (see page 161). This leads to models in which possible computations are assumed, say, to correspond to combinations of differential equations—as in Claude Shannon's 1941 general-purpose analog computer. And if one assumes—as is usually implicitly done in traditional mathematics—that any solutions that exist to these equations can somehow always be found then at least in principle this allows computations impossible for discrete systems to be done.

■ **Initial conditions.** Traditional mathematics tends to assume that real numbers with absolutely any digit sequence can be set up. And if this were the case, then the digits of an initial condition could for example be the table for an oracle of the kind discussed on page 1126—and even a simple shift mapping could then yield output that is computationally more sophisticated than any standard discrete system. But just as in my discussion of chaos theory in Chapter 7, any reasonably complete theory must address how such an initial condition could have been constructed. And presumably the only way is to have another system that already violates the Principle of Computational Equivalence.

■ **Constructible reals.** Instead of finding successive digits using systems like Turing machines, one can imagine constructing complete real numbers using idealizations of mechanical processes. An example studied since antiquity involves finding lengths or angles using a ruler and compass (i.e. as intersections between lines and circles). However, as was shown in the 1800s, this method can yield only numbers formed by operating on rationals with combinations of *Plus*, *Times* and *Sqrt*. (Thus it is impossible with ruler and compass to construct π and "square the circle" but it is possible to construct 17-gons or other n-gons for which *FunctionExpand[Sin[π/n]]* contains only *Plus*, *Times* and *Sqrt*.) Linkages consisting of rods of integer lengths always trace out algebraic curves (or algebraic surfaces in 3D) and in general allow any algebraic number (as represented by *Root*) to be constructed. (Linkages were used by the late 1800s not only in machines such as steam engines, but also in devices for analog computation. More recently they have appeared in robotics.) Note that above degree 4, algebraic numbers cannot in general be expressed in radicals involving only *Plus*, *Times* and *Power* (see page 945).

■ **Page 732 · Equations.** For any purely algebraic equation involving real numbers it is possible to find a bound on the size of any isolated solutions it has, and then to home in on their actual values. But as discussed on page 786, nothing similar is true for equations involving only integers, and in this case finding solutions can in effect require following the evolution of a system like a cellular automaton for infinitely many steps. If one allows trigonometric functions, any equation for integers can be converted to one for real numbers; for example $x^2 + y^2 == z^2$ for integers is equivalent to $Sin[\pi x]^2 + Sin[\pi y]^2 + Sin[\pi z]^2 + (x^2 + y^2 - z^2)^2 == 0$ for real numbers.

■ **Page 732 · ODEs.** The method of compressing time using algebraic transformations works not only in partial but also in ordinary differential equations.

■ **Emulating discrete systems.** Despite it often being assumed that continuous systems are computationally more sophisticated than discrete ones, it has in practice proved surprisingly difficult to make continuous systems emulate discrete ones. Some integer functions can readily be obtained by supplying integer arguments to continuous functions, so that for example *Mod[x, 2]* corresponds to $Sin[\pi x/2]^2$ or *(1 - Cos[π x])/2*,

```
Mod[x, 3] ↔ 1 + 2/3 (Cos[2/3 π (x - 2)] - Cos[2 π x/3])

Mod[x, 4] ↔ (3 - 2 Cos[π x/2] - Cos[π x] - 2 Sin[π x/2])/2

Mod[x, n] ↔ Sum[j Product[(Sin[π (x - i - j)/n]/
                Sin[π i/n])^2, {i, n - 1}], {j, n - 1}]
```

(As another example, *If[x > 0, 1, 0]* corresponds to *1 - 1/Gamma[1 - x]*.) And in this way the discrete system $x \to$ *If[EvenQ[x], 3 x/2, 3 (x + 1)/2]* from page 122 can be emulated by the continuous iterated map $x \to$ *(3 + 6 x - 3 Cos[π x])/4*. This approach can then be applied to the universal arithmetic system on page 673, establishing that continuous iterated maps can in principle emulate discrete universal systems. A similar result presumably holds for ordinary and therefore also partial differential equations (PDEs). One might expect, however, that it should be possible to construct a PDE that quite directly emulates a system like a cellular automaton. And to do this approximately is not difficult. For as suggested by the bottom row of pictures on page 732 one can imagine having localized structures whose interactions emulate the rules of the cellular automaton. And one can set things up so that these structures exhibit the analog of attractors, and evolve towards one of a few discrete states. But the problem is that

in finite time one cannot expect that they will precisely reach such states. (This is somewhat analogous to the issue of asymptotic particle states in the foundations of quantum field theory.) And this means that the overall state of the system will not be properly prepared for the next step of cellular automaton evolution.

Generating repetitive patterns with continuous systems is straightforward, but generating even nested ones is not. Page 147 showed how $Sin[x] + Sin[\sqrt{2}\, x]$ has nested features, and these are reflected in the distribution of eigenvalues for ODEs containing such functions. Strange attractors for many continuous systems also show various forms of Cantor sets and nesting.

▪ **Page 732 · Time and gravity.** General relativity implies that time can be affected by gravitational fields—and that for example a process in a lower gravitational field will seem to be going faster if it is looked at by an observer in a higher gravitational field. (Related phenomena associated with motion in special relativity are more difficult to interpret in a static way.) But presumably there are effects that prevent infinite speedups. For if, say, energy were coming from a process at a constant rate, then an infinite speedup would lead to infinite energy density, and thus presumably to infinite gravitational fields that would change the system.

At least formally, general relativity does nevertheless suggest infinite transformations of time in various cases. For example, to a distant observer, an object falling into a black hole will seem to take an infinite time to cross the event horizon—even though to the object itself only a finite time will seem to have passed. One might have thought that this would imply in reverse that to an observer moving with the object the whole infinite future of the outside universe would in effect seem to go by in a finite time. But in the simplest case of a non-rotating black hole (Schwarzschild metric), it turns out that an object will always hit the singularity at the center before this can happen. In a rotating but perfectly spherical black hole (Kerr metric), the situation is nevertheless different, and in this case the whole infinite future of the outside universe can indeed in principle be seen in the finite time between crossing the outer and inner event horizons. But for the reasons mentioned above, this very fact presumably implies instability, and the whole effect disappears if there is any deviation from perfect spherical symmetry.

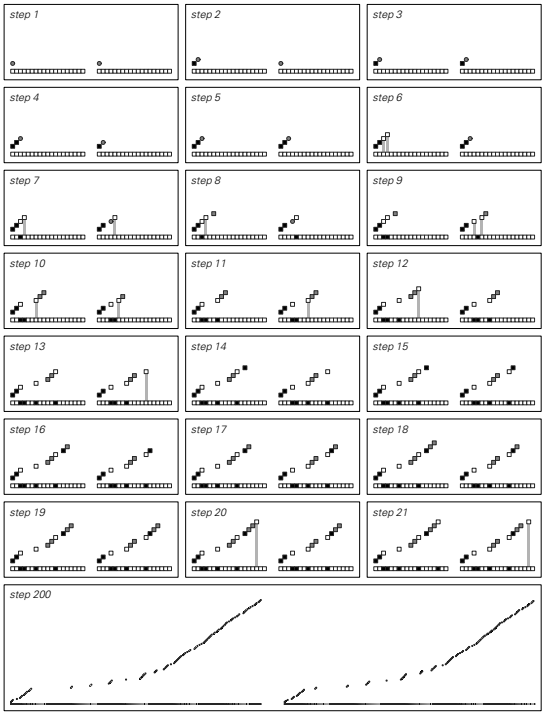
Even without general relativity there are already issues with time and gravity. For example, it was shown in 1990 that close encounters in a system of 5 idealized point masses can lead to infinite accelerations which cause one mass to be able to go infinitely far in a finite time.

▪ **Page 733 · Human thinking.** The discovery in this book that even extremely simple programs can give rise to behavior vastly more complex than expected casts suspicion on any claim that programs are fundamentally unable to reproduce features of human thinking. But complete evidence that human thinking follows the Principle of Computational Equivalence will presumably come only gradually as practical computer systems manage to emulate more and more aspects of human thinking. (See page 628.)

▪ **Page 734 · Intermediate degrees.** As discussed on page 753, an important indication of computational sophistication in a system is for its ultimate behavior to be undecidable, in the sense that a limited number of steps in a standard universal system cannot determine in general what the system will do after an infinite number of steps, and whether, for example, it will ever in some sense halt. Such undecidability is inevitable in any system that is universal. But what about other systems? So long as one only ever looks at the original input and final output it turns out that one can construct a system that exhibits undecidability but is not universal. One trivial way to do so is to take a universal system but modify it so that if it ever halts its output is discarded and, say, replaced by its original input. The lack of meaningful output prevents such a system from being universal, but the question of whether the system halts is still undecidable. Nevertheless, the pattern of this undecidability is just the same as for the underlying universal system. So one can then ask whether it is possible to have a system which exhibits undecidability, but with a pattern that does not correspond to that of any universal system.

As I discuss on page 1137, almost all known proofs of undecidability in practice work by reduction to the halting problem for some universal system—this is, by showing that if one could resolve whatever is supposed to be undecidable then one could also solve the halting problem for a universal system. But in 1956 Richard Friedberg and Albert Muchnik both gave an intricate and abstract construction of a system that has a halting problem which is undecidable but is not reducible to the halting problem of any universal system.

The pictures at the top of the facing page show successive steps in the evolution of an analog of their system. The input is an integer that gives a position in either of the two rows of cells at the bottom of each picture. All these cells are initially white, but some eventually become black—and the system is considered to halt for a particular input if the corresponding cell ever becomes black.



The rules for the system are quite complicated, and in essence work by progressively implementing a generalization of a diagonal argument of the kind discussed on page 1128. Note first that the configuration of cells in the rows at the bottom of each picture can be thought of as successive finite approximations to tables for an oracle (see page 1126) which gives the solution to the halting problem for each possible input to the system. To set up the generalized diagonal argument one needs a way to list all possible programs. Any type of program that supports universality can be used for this purpose; the pictures shown use essentially the register machines from page 97. Each row above the bottom one corresponds in effect to a successive register machine—and shows, if relevant, its output when given as input the integer corresponding to that position in the row, together with the complete bottom row of cells found so far. (A dot indicates that the register machine does not halt.) The way the system works is to put down new black cells in the bottom row in just such a way as to arrange that for any register machine at least the output shown will ultimately not agree with the cells in the bottom row. As indicated by vertical gray lines, there is sometimes temporary agreement, but this is always removed within a finite number of steps.

The fact that no register machine can ever ultimately give output that agrees everywhere with the bottom row of cells then demonstrates that the halting problem for the system—whose results appear in the bottom row—must be undecidable. Yet if this halting problem were reducible to a halting problem for a universal system, then by using its results one should ultimately be able to solve the halting problem for any system. However, even using the complete bottom row of cells on the left it turns out that the construction is such that no register machine can ever yield results after any finite number of steps that agree everywhere with the row of cells on the right—thus demonstrating that the halting problem for the system is not reducible to the halting problem for a universal system.

Note however that this result is extremely specific to looking only at what is considered output from the system, and that inside the system there are all sorts of components that are definitely universal.

Explaining the Phenomenon of Complexity

▪ **Definition of complexity.** See page 557.

▪ **Ingredients for complexity.** With its emphasis on breaking systems down to find their underlying elements traditional science tends to make one think that any important overall property of a system must be a consequence of some specific feature of its underlying construction. But the results of this section imply that for complexity this is not the case. For as discussed on page 1126 there is no direct structural criterion for sophisticated computation and universality. And indeed most ways of ensuring that these do not occur are in essence equivalent just to saying that the overall behavior exhibits some specific regularity and is therefore not complex.

▪ **Relativism and equivalence.** Although the notion has been discussed since antiquity, it has become particularly common in the academic humanities in the past few decades to believe that there can be no valid absolute conclusions about the world—only statements made relative to particular cultural contexts. My emphasis of the importance of perception and analysis might seem to support this view, and to some extent it does. But the Principle of Computational Equivalence implies that in the end essentially any method of perception and analysis that can actually be implemented in our universe must have a certain computational equivalence, and must therefore at least in some respects come to the same absolute conclusions.

Computational Irreducibility

■ **History.** The notion that there could be fundamental limits to knowledge or predictability has been discussed repeatedly since antiquity. But most often it has been assumed that the origin of this must be inadequacy in models, not difficulty in working out their consequences. And indeed already in the 1500s with the introduction of symbolic algebra and the discovery of formulas for solving cubic and quartic equations the expectation began to develop that with sufficient cleverness it should be possible to derive a formula for the solution to any purely mathematical problem. Infinitesimals were sometimes thought to get in the way of finite understanding—but this was believed to be overcome by calculus. And when mathematical models for natural systems became widespread in the late 1600s it was generally assumed that their basic consequences could always be found in terms of formulas or geometrical theorems, perhaps with fairly straightforward numerical calculations required for connection to practical situations. In discussing gravitational interactions between many planets Isaac Newton did however comment in 1684 that "to define these motions by exact laws admitting of easy calculation exceeds, if I am not mistaken, the force of any human mind". But in the course of the 1700s and 1800s formulas were successfully found for solutions to a great many problems in mathematical physics (see note below)—at least when suitable special functions (see page 1091) were introduced. The three-body problem (see page 972) nevertheless continued to resist efforts at general solution. In the 1820s it was shown that quintic equations cannot in general be solved in terms of radicals (see page 1137), and by the 1890s it was known that degree 7 equations cannot in general be solved even if elliptic functions are allowed. Around 1890 it was then shown that the three-body problem could not be solved in general in terms of ordinary algebraic functions and integrals (see page 972). However, perhaps in part because of a shift towards probabilistic theories such as quantum and statistical mechanics there remained the conviction that for relevant aspects of behavior formulas should still exist. The difficulty for example of finding more than a few exact solutions to the equations of general relativity was noted—but a steady stream of results (see note below) maintained the belief that with sufficient cleverness a formula could be found for behavior according to any model.

In the 1950s computers began to be used to work out numerical solutions to equations—but this was seen mostly as a convenience for applications, not as a reflection of any basic necessity. A few computer experiments were done on systems with simple underlying rules, but partly because Monte Carlo methods were sometimes used, it was typically assumed that their results were just approximations to what could in principle be represented by exact formulas. And this view was strengthened in the 1960s when solitons given by simple formulas were found in some of these systems.

The difficulty of solving equations for numerical weather prediction was noted even in the 1920s. And by the 1950s and 1960s the question of whether computer calculations would be able to outrun actual weather was often discussed. But it was normally assumed that the issue was just getting a better approximation to the underlying equations—or better initial measurements—not something more fundamental.

Particularly in the context of game theory and cybernetics the idea had developed in the 1940s that it should be possible to make mathematical predictions even about complex human situations. And for example starting in the early 1950s government control of economies based on predictions from linear models became common. By the early 1970s, however, such approaches were generally seen as unsuccessful, but it was usually assumed that the reason was not fundamental, but was just that there were too many disparate elements to handle in practice.

The notions of universality and undecidability that underlie computational irreducibility emerged in the 1930s, but they were not seen as relevant to questions arising in natural science. Starting in the 1940s they were presumably the basis for a few arguments made about free will and fundamental unpredictability of human behavior (see page 1135), particularly in the context of economics. And in the late 1950s there was brief interest among philosophers in connecting results like Gödel's Theorem to questions of determinism—though mostly there was just confusion centered around the difficulty of finding countable proofs for statements about the continuous processes assumed to occur in physics.

The development of algorithmic information theory in the 1960s led to discussion of objects whose information content cannot be compressed or derived from anything shorter. But as indicated on page 1067 this is rather different from what I call computational irreducibility. In the 1970s computational complexity theory began to address questions about overall resources needed to perform computations, but concentrated on computations that perform fairly specific known practical tasks. At the beginning of the 1980s, however, it was noted that certain problems about models of spin glasses were NP-complete. But there was no immediate realization that this was connected to any underlying general phenomenon.

Starting in the late 1970s there was increasing interest in issues of predictability in models of physical systems. And it

was emphasized that when the equations in such models are nonlinear it often becomes difficult to find their solutions. But usually this was at some level assumed to be associated with sensitive dependence on initial conditions and the chaos phenomenon—even though as we saw on page 1098 this alone does not even prevent there from being formulas.

By the early 1980s it had become popular to use computers to study various models of natural systems. Sometimes the idea was to simulate a large collection of disparate elements, say as involved in a nuclear explosion. Sometimes instead the idea was to get a numerical approximation to some fairly simple partial differential equation, say for fluid flow. Sometimes the idea was to use randomized methods to get a statistical approximation to properties say of spin systems or lattice gauge theories. And sometimes the idea was to work out terms in a symbolic perturbation series approximation, say in quantum field theory or celestial mechanics. With any of these approaches huge amounts of computer time were often used. But it was almost always implicitly assumed that this was necessary in order to overcome the approximations being used, and not for some more fundamental reason.

Particularly in physics, there has been some awareness of examples such as quark confinement in QCD where it seems especially difficult to deduce the consequences of a theory—but no general significance has been attached to this.

When I started studying cellular automata in the early 1980s I was quickly struck by the difficulty of finding formulas for their behavior. In traditional models based for example on continuous numbers or approximations to them there was usually no obvious correspondence between a model and computations that might be done about it. But the evolution of a cellular automaton was immediately reminiscent of other computational processes—leading me by 1984 to formulate explicitly the concept of computational irreducibility.

No doubt an important reason computational irreducibility was not identified before is that for more than two centuries students had been led to think that basic theoretical science could somehow always be done with convenient formulas. For almost all textbooks tend to discuss only those cases that happen to come out this way. Starting in earnest in the 1990s, however, the influence of *Mathematica* has gradually led to broader ranges of examples. But there still remains a very widespread belief that if a theoretical result about the behavior of a system is truly fundamental then it must be possible to state it in terms of a simple mathematical formula.

■ **Exact solutions.** Some notable cases where closed-form analytical results have been found in terms of standard mathematical functions include: quadratic equations (~2000 BC) (*Sqrt*); cubic, quartic equations (1530s) ($x^{1/n}$); 2-body problem (1687) (*Cos*); catenary (1690) (*Cosh*); brachistochrone (1696) (*Sin*); spinning top (1849; 1888; 1888) (*JacobiSN*; *WeierstrassP*; hyperelliptic functions); quintic equations (1858) (*EllipticTheta*); half-plane diffraction (1896) (*FresnelC*); Mie scattering (1908) (*BesselJ*, *BesselY*, *LegendreP*); Einstein equations (Schwarzschild (1916), Reissner-Nordström (1916), Kerr (1963) solutions) (rational and trigonometric functions); quantum hydrogen atom and harmonic oscillator (1927) (*LaguerreL*, *HermiteH*); 2D Ising model (1944) (*Sinh*, *EllipticK*); various Feynman diagrams (1960s–1980s) (*PolyLog*); KdV equation (1967) (*Sech* etc.); Toda lattice (1967) (*Sech*); six-vertex spin model (1967) (*Sinh* integrals); Calogero-Moser model (1971) (*Hypergeometric1F1*); Yang-Mills instantons (1975) (rational functions); hard-hexagon spin model (1979) (*EllipticTheta*); additive cellular automata (1984) (*MultiplicativeOrder*); Seiberg-Witten supersymmetric theory (1994) (*Hypergeometric2F1*). When problems are originally stated as differential equations, results in terms of integrals ("quadrature") are sometimes considered exact solutions—as occasionally are convergent series. When one exact solution is found, there often end up being a whole family—with much investigation going into the symmetries that relate them. It is notable that when many of the examples above were discovered they were at first expected to have broad significance in their fields. But the fact that few actually did can be seen as further evidence of how narrow the scope of computational reducibility usually is. Notable examples of systems that have been much investigated, but where no exact solutions have been found include the 3D Ising model, quantum anharmonic oscillator and quantum helium atom.

■ **Amount of computation.** Computational irreducibility suggests that it might be possible to define "amount of computation" as an independently meaningful quantity—perhaps vaguely like entropy or amount of information. And such a quantity might satisfy laws vaguely analogous to the laws of thermodynamics that would for example determine what processes are possible and what are not. If one knew the fundamental rules for the universe then one way in principle to define the amount of computation associated with a given process would be to find the minimum number of applications of the rules for the universe that are needed to reproduce the process at some level of description.

■ **Page 743 · More complicated rules.** The standard rule for a cellular automaton specifies how every possible block of cells of a certain size should be updated at every step. One can imagine finding the outcome of evolution more efficiently by adding rules that specify what happens to larger blocks of cells after more steps. And as a practical matter, one can look

up different blocks using a method like hashing. But much as one would expect from data compression this will only in the end work more efficiently if there are some large blocks that are sufficiently common. Note that dealing with blocks of different sizes requires going beyond an ordinary cellular automaton rule. But in a sequential substitution system—and especially in a multiway system (see page 776)—this can be done just as part of an ordinary rule.

■ **Page 744 · Reducible systems.** The color of a cell at step t and position x can be found by starting with initial condition

```
Flatten[With[{w = Max[Ceiling[Log[2, {t, x}]]]},
    {2 Reverse[IntegerDigits[t, 2, w]] + 1,
      5, 2 IntegerDigits[x, 2, w] + 2}]]
```

then for rule 188 running the cellular automaton with rule

```
{{a : (1 | 3), 1 | 3, _} → a, {_, 2 | 4, a : (2 | 4)} → a,
  {3, 5 | 10, 2} → 6, {1, 5 | 7, 4} → 0, {3, 5, 4} → 7,
  {1, 6, 2} → 10, {1, 6 | 11, 4} → 8, {3, 6 | 8 | 10 | 11, 4} → 9,
  {3, 7 | 9, 2} → 11, {1, 8 | 11, 2} → 9, {3, 11, 2} → 8,
  {1, 9 | 10, 4} → 11, {_, a_ /; a > 4, _} → a, {_, _, _} → 0}
```

and for rule 60 running the cellular automaton with rule

```
{{a : (1 | 3), 1 | 3, _} → a, {_, 2 | 4, a : (2 | 4)} → a,
  {1, 5, 4} → 0, {_, 5, _} → 5, {_, _, _} → 0}
```

■ **Speed-up theorems.** That there exist computations that are arbitrarily computationally reducible was noted in work on the theory of computation in the mid-1960s.

■ **Page 745 · Mathematical functions.** The number of bit operations needed to add two n-digit numbers is of order n. The number of operations $m[n]$ needed to multiply them increases just slightly more rapidly than n (see page 1093). (Even if one can do operations on all digits in parallel it still takes of order n steps in a system like a cellular automaton for the effects of different digits to mix together—though see also page 1149.) The number of operations to evaluate *Mod[a, b]* is of order n if a has n digits and b is small. Many standard continuous mathematical functions just increase or decrease smoothly at large x (see page 917). The main issue in evaluating those that exhibit regular oscillations at large x is to find their oscillation period with sufficient precision. Thus for example if x is an integer with n digits then evaluating *Sin[x]* or *FractionalPart[x c]* requires respectively finding π or c to n-digit precision. It is known how to evaluate π (see page 912) and all standard elementary functions to n-digit precision using about *Log[n] m[n]* operations. (This can be done by repeatedly making use of functional relations such as $Exp[2x] == Exp[x]^2$ which express $f[2x]$ as a polynomial in $f[x]$; such an approach is known to work for elementary, elliptic, modular and other functions associated with *ArithmeticGeometricMean* and for example *DedekindEta*.) Known methods for high-precision evaluation of special functions—usually based in the end on series representations—typically require of order $n^{1/s}\, m[n]$ operations, where s is often 2 or 3. (Examples of more difficult cases include *HypergeometricPFQ[a, b, 1]* and *StieltjesGamma[k]*, where logarithmic series can require an exponential number of terms. Evaluation of *BernoulliB[x]* is also difficult.) Any iterative procedure (such as *FindRoot*) that yields a constant multiple more digits at each step will take about *Log[n]* steps to get n digits. Roots of polynomials can thus almost always be found with *NSolve* in about *Log[n] m[n]* operations. If one evaluates *NIntegrate* or *NDSolve* by effectively fitting functions to order s polynomials the difficulty of getting results with n-digit precision typically increases like $2^{n/s}$. An adaptive algorithm such as Romberg integration reduces this to about $2\hat{}\sqrt{n}$. The best-known algorithms for evaluating *Zeta[1/2 + I x]* (see page 918) to fixed precision take roughly $\sqrt{x}$ operations—or $2^{n/2}$ operations if x is an n-digit integer. (The evaluation is based on the Riemann-Siegel formula, which involves sums of about $\sqrt{x}$ cosines.) Unlike for continuous mathematical functions, known algorithms for number theoretical functions such as *FactorInteger[x]* or *MoebiusMu[x]* typically seem to require a number of operations that grows faster with the number of digits n in x than any power of n (see page 1090).

■ **Formulas.** It is always in principle possible to build up some kind of formula for the outcome of any process of evolution, say of a cellular automaton (see page 618). But for there to be computational reducibility this formula needs to be simple and easy to evaluate—as it is if it consists just of a few standard mathematical functions (see note above; page 1098).

■ **Page 747 · Short computations.** Some properties include:

(a) The regions are bounded by the hyperbolas $x\,y == Exp[n/2]$ for successive integers n.

(d) There is approximate repetition associated with rational approximations to π (for example with period 22), but never precise repetition.

(e) The pattern essentially shows which x are divisors of y, just as on pages 132 and 909.

(h) $Mod[Quotient[s, 2^n], 2]$ extracts the digit associated with 2^n in the base 2 digit sequence of s.

(i) Like (e), except that colors at neighboring positions alternate.

(l) See page 613.

(m) The pattern can be generated by a 2D substitution system with rule *{1 -> {{0, 0}, {0, 1}}, 0 -> {{1, 1}, {1, 0}}}* (see page 583).

(See also page 870.)

Even though standard mathematical functions are used, few of the pictures can readily be generalized to continuous values of x and y.

▪ **Intrinsic limits in science.** Before computational irreducibility other sources of limits to science that have been discussed include: measurement in quantum mechanics, prediction in chaos theory and singularities in gravitation theory. As it happens, in each of these cases I suspect that the supposed limits are actually just associated with a lack of correct analysis of all elements of the relevant systems. In mathematics, however, more valid intrinsic limits—much closer to computational irreducibility—follow for example from Gödel's Theorem.

The Phenomenon of Free Will

▪ **History.** Early in history it seems to have generally been assumed that everything about humans must ultimately be determined by unchangeable fate—which it was sometimes thought could be foretold by astrology or other forms of divination. Most Greek philosophers seem to have believed that their various mechanical or moral theories implied rigid determination of human actions. But especially with the advent of the Christian religion the notion that humans can at some level make free choices—particularly about whether to do good or not—emerged as a foundational idea. (The idea had also arisen in Persian and Hebrew religions and legal systems, and was supported by Roman lawyers such as Cicero.) How this could be consistent with God having infinite power was not clear, although around 420 AD Augustine suggested that while God might have infinite knowledge of the future we as humans could not—yielding what can be viewed as a very rough analog of my explanation for free will. In the 1500s some early Protestants made theological arguments against free will—and indeed issues of free will remain a feature of controversy between Christian denominations even today.

In the mid-1600s philosophers such as Thomas Hobbes asserted that minds operate according to definite mechanisms and therefore cannot exhibit free will. In the late 1700s philosophers such as Immanuel Kant—agreeing with earlier work by Gottfried Leibniz—claimed instead that at least some parts of our minds are free and not determined by definite laws. But soon thereafter scientists like Pierre-Simon Laplace began to argue for determinism throughout the universe based on mathematical laws. And with the increasing success of science in the 1800s it came to be widely believed that there must be definite laws for all human actions—providing a foundation for the development of psychology and the social sciences.

In the early 1900s historians and economists emphasized that there were at least not simple laws for various aspects of human behavior. But it was nevertheless typically assumed that methods based on physics would eventually yield deterministic laws for human behavior—and this was for example part of the inspiration for the behaviorist movement in psychology in the mid-1900s. The advent of quantum mechanics in the 1920s, however, showed that even physics might not be entirely deterministic—and by the 1940s the possibility that this might lead to human free will was being discussed by physicists, philosophers and historians. Around this time Karl Popper used both quantum mechanics and sensitive dependence on initial conditions (see also page 971) to argue for fundamental indeterminism. And also around this time Friedrich Hayek (following ideas of Ludwig Mises in the early 1900s) suggested—presumably influenced by work in mathematical logic—that human behavior might be fundamentally unpredictable because in effect brains can explain only systems simpler than themselves, and can thus never explain their own operation. But while this has some similarity to the ideas of computational irreducibility in this book it appears never to have been widely studied.

Questions of free will and responsibility have been widely discussed in criminal and other law since at least the 1800s (see note below). In the 1960s and 1970s ideas from popular psychology tended to diminish the importance of free will relative to physiology or environment and experiences. In the 1980s, however, free will was increasingly attributed to animals other than humans. Free will for computers and robots was discussed in the 1950s in science fiction and to some extent in the field of cybernetics. But following lack of success in artificial intelligence it has for the most part not been seriously studied. Sometimes it is claimed that Gödel's Theorem shows that humans cannot follow definite rules—but I argue on page 1158 that this is not correct.

▪ **Determinism in brains.** Early investigations of internal functioning in the brain tended to suggest considerable randomness—say in the sequence of electrical pulses from a nerve cell. But in recent years, with more extensive measurement methods, there has been increasing evidence for precise deterministic underlying rules. (See pages 976 and 1011.)

▪ **Amounts of free will.** In my theory the amount of free will associated with a particular decision is in effect related to the amount of computation required to arrive at it. In conscious thinking we can to some extent scrutinize the processes we

use, and assess how much computation they involve. But in unconscious thinking we cannot. And probably often these just involve memory lookups with rather little computation. But other unconscious abilities like intuition presumably involve more sophisticated computation.

▪ **Responsibility.** It is often assumed that if there are definite underlying rules for our brains then it cannot be meaningful to say that we have any ultimate moral or legal responsibility for our actions. For traditional ideas lead to the notion that in this case all our actions must somehow be thought of as the direct result of whatever external causes (over which we have no control) are responsible for the underlying rules in our brains and the environment in which we find ourselves. But if the processes in our brains are computationally irreducible then as discussed in the main text their outcome can seem in many respects free of underlying rules, making it reasonable to view the processes themselves as what is really responsible for our actions. And since these processes are intrinsic to us, it makes sense to treat us as responsible for their effects.

Several different theories are used in practical legal systems. The theory popular from the behavioral sciences tends to assume that human actions can be understood from underlying rules for the brain, and that people should be dealt with according to the rules they have—which can perhaps be modified by some form of treatment. But computational irreducibility can make it essentially impossible to find what general behavior will arise from particular rules—making it difficult to apply this theory. The alternative pragmatic theory popular in rational philosophy and economics suggests that behavior in legal matters is determined through calculations based on laws and the deterrents they provide. But here there is the issue that computational irreducibility can make it impossible to foresee what consequences a given law will have. Western systems of law tend to be dominated by the moral theory that people should somehow get what they deserve for choices they made with free will—and my explanation now makes this consistent with the existence of definite underlying rules for the brain.

Young children, animals and the insane are typically held less responsible for their actions. And in a moral theory of law this can be understood in my approach as a consequence of the computations they do being less sophisticated—so that their outcome is less free of the environment and of their underlying rules. (In a pragmatic theory the explanation would presumably be that less sophisticated computations would not be up to the task of handling the elaborate system of incentives that laws had defined.)

▪ **Will and purpose.** Things that are too predictable do not normally seem free. But things that are too random also do not normally seem to be associated with the exercise of a will. Thus for example continual random twitching in our muscles is not normally thought to be a matter of human will, even though some of it is the result of signals from our brains. For typically one imagines that if something is to be a genuine reflection of human will then there must be some purpose to it. In general it is very difficult to assess whether something has a purpose (see page 829). But in capturing the most obvious aspects of human will what seems to be most important is at least short-term coherence and consistency of action—as often exists in class 4, but not class 3, systems.

▪ **Source of will.** Damage to a human brain can lead to apparent disappearance of the will to act, and there is some evidence that one small part of the brain is what is crucial.

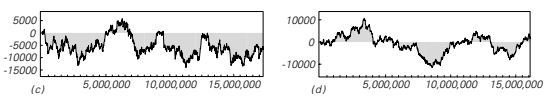
Undecidability and Intractability

▪ **History.** In the early 1900s, particularly in the context of the ideas of David Hilbert, it was commonly believed that there should be a finite procedure to decide the truth of any mathematical statement. That this is not the case in the standard theory of arithmetic was in effect established by Kurt Gödel in 1931 (see page 1158). Alonzo Church gave the first explicit example of an undecidable problem in 1935 when he showed that no finite procedure in lambda calculus could guarantee to determine the equivalence of two lambda expressions. (A corollary to Gödel's proof had in fact already supplied another explicit undecidable problem by implying that no finite procedure based on recursive functions could decide whether a given primitive recursive function is identically 0.) In 1936 Alan Turing then showed that the halting problem for Turing machines could not be solved in general in a finite number of steps by any Turing machine. Some similar issues had already been considered by Emil Post in the context of tag and multiway systems starting in the 1920s, and in 1947 Post and Andrei Markov were able to establish that an existing mathematical question—the word problem for semigroups (see page 1141)—was undecidable. By the 1960s undecidability was being found in all sorts of systems, but most of the examples were too complicated to seem of much relevance in practical mathematics or computing. And apart from a few vague mentions in fields like psychology, undecidability was viewed mainly as a highly abstract curiosity of no importance to ordinary science. But in the early 1980s my experiments on cellular automata convinced me that undecidability is vastly more common than had been assumed, and in my 1984 paper

"Undecidability and intractability in theoretical physics" I argued that it should be important in many issues in physics and elsewhere.

▪ **Mathematical impossibilities.** It is sometimes said that in the 1800s problems such as trisecting angles, squaring the circle, solving quintics, and integrating functions like $Exp[x^2]$ were proved mathematically impossible. But what was actually done was just to show that these problems could not be solved in terms of particular levels of mathematical constructs—say square roots (as in ruler and compass constructions discussed on page 1129), arbitrary roots, or elementary transcendental functions. And in each case higher mathematical constructs that seem in some sense no less implementable immediately allow the problems to be solved. Yet with undecidability one believes that there is absolutely no construct that can explicitly exist in our universe that allows the problem to be solved in any finite way. And unlike traditional mathematical impossibilities, undecidability is normally formulated purely in terms of ordinary integers—making it in a sense necessary to collapse basic distinctions between finite and infinite quantities if any higher-level constructs are to be included.

▪ **Page 755 · Code 1004600.** In cases (c) and (d) steady growth at about 0.035 and 0.039 cells per step (of which 28% on average are non-white) is seen up to at least 20 million steps, though there continue to be fluctuations as shown below.



▪ **Halting problems.** A classic example of a problem that is known in general to be undecidable is whether a given Turing machine will ever halt when started from a given initial condition. Halting is usually defined by the head of the Turing machine reaching a special halt state. But other criteria can equally well be used—say the head reaching a particular position (see page 759), or a certain pattern of colors being formed on the tape. And in a system like a cellular automaton a halting problem can be set up by asking whether a cell at a particular position ever turns a particular color, or whether, more globally, the complete state of the system ever reaches a fixed point and no longer changes.

In practical computing, one usually thinks of computational programs as being set up much like the register machines of page 896 and halting when they have finished executing their instructions. User interface and operating system programs are not normally intended to halt in an explicit sense, although without external input they often reach states that do not change. *Mathematica* works by taking its input and repeatedly applying transformation rules—a process which normally reaches a fixed point that is returned as the answer, but with definitions like $x = x + 1$ (x having no value) formally does not.

▪ **Proofs of undecidability.** Essentially the same argument due to Alan Turing used on page 1128 to show that most numbers cannot be computable can also be used to show that most problems cannot be decidable. For a problem can be thought of as an infinite list of solutions for successive possible inputs. But this is analogous to a digit sequence of a real number. And since any program for a universal system can be specified by an integer it follows that there must be many problems for which no such program can be given.

To show that a particular problem like the halting problem is undecidable one typically argues by contradiction, setting up analogs of self-referential logic paradoxes such as "this statement is false". Suppose that one had a Turing machine m that could solve the halting problem, in the sense that it itself would always halt after a finite number of steps, but it would determine whether any Turing machine whose description it was given as input would ever halt. One way to see that this is not possible is to imagine modifying m to make a machine m' that halts if its input corresponds to a machine that does not halt, but otherwise goes into an infinite loop and does not itself halt. For if one considers feeding m' as input to itself there is immediately no consistent answer to the question of whether m' halts—leading to the conclusion that in fact no machine m could ever exist in the first place. (To make the proof rigorous one must add another level of self-reference, say setting up m' to ask m whether a Turing machine will halt when fed its own description as input.) In the main text I argued that undecidability is a consequence of universality. In the proof above universality is what guarantees that any Turing machine can successfully be described in a way that can be fed as input to another Turing machine.

▪ **Page 756 · Examples of undecidability.** Once universality exists in a system it is known from Gordon Rice's 1953 theorem and its generalizations that most questions about ultimate behavior will be undecidable unless their answers are always trivially the same. Undecidability has been demonstrated in various seemingly rather different types of systems, most often by reduction to halting (termination) problems for multiway systems.

In formal language theory, questions about regular languages are always decidable, but ones about context-free languages (see page 1103) are already often not. It is decidable whether

such a language is finite, but not whether it contains every possible string, is regular, is unambiguous, or is equivalent to a language with a different grammar.

In mathematical logic, it can be undecidable whether statements are provable from a given axiom system—say predicate logic or Peano arithmetic (see page 782). It is also undecidable whether one axiom system is equivalent to another—even for basic logic (see page 1170).

In algebra and related areas of mathematics problems of equivalence between objects built up from elements that satisfy relations are often in general undecidable. Examples are word problems for groups and semigroups (see page 1141), and equivalence of finitely specified 4D manifolds (see page 1051). (Equivalence for 3D manifolds is thought to be decidable.) A related undecidable problem is whether two integer matrices can be multiplied together in some sequence to yield the zero matrix. It is also undecidable whether two sets of relations specify the same group or semigroup.

In combinatorics it is known in general to be undecidable whether a given set of tiles can cover the plane (see page 1139). And from this follows the undecidability of various problems about 2D cellular automata (see note below) and spin systems. Also undecidable are many questions about whether strings exist that satisfy particular constraints (see below).

In number theory it is known to be undecidable whether Diophantine equations have solutions (i.e. whether algebraic equations have integer solutions) (see page 786). And this means for example that it is in general undecidable whether expressions that involve both algebraic and trigonometric functions can be zero for real values of variables, or what the values of integrals are in which such expressions appear as denominators (compare page 916).

In computer science, general problems about verifying the possible behavior of programs tend to be undecidable, usually being directly related to halting problems. It is also for example undecidable whether a given program is the shortest one that produces particular output (see page 1067).

It is in general undecidable whether a given system exhibits universality—or undecidability.

▪ **Undecidability in cellular automata.** For 1D cellular automata, almost all questions about ultimate limiting behavior are undecidable, even ones that ask about average properties such as density and entropy. (This results in undecidability in classification schemes, as mentioned on page 948.) Questions about behavior after a finite number of steps, even with infinite initial conditions, tend to be decidable for 1D cellular automata, and related to regular languages (see page 957). In 2D cellular automata, however, even questions about a single step are often undecidable. Examples include whether any configurations are invariant under the cellular automaton evolution (see page 942), and, as established by Jarkko Kari in the late 1980s, whether the evolution is reversible, or can generate every possible configuration (see page 959).

▪ **Natural systems.** Undecidable questions arise even in some traditional classes of models for natural systems. For example, in a generalized Ising model (see page 944) for a spin system the undecidability of the tiling problem implies that it is undecidable whether a given energy function leads to a phase transition in the infinite size limit. Somewhat similarly, the undecidability of equivalence of 4-manifolds implies undecidability of questions about quantum gravity models. In models based both on equations and other kinds of rules the existence of formulas for conserved quantities is in general undecidable. In models that involve continuous quantities it can be more difficult to formulate undecidability. But I strongly suspect that with appropriate definitions there is often undecidability in for example the three-body problem, so that the questions such as whether one of the bodies in a particular scattering process will ever escape to infinity are in general undecidable. In biology formal models for neural processes often involve undecidability, so that in principle it can be undecidable whether, say, there is any particular stimulus that will lead to a given response. Formal models for morphogenesis can also involve undecidability, so that for example it can in principle be undecidable whether a particular organism will ever stop growing, or whether a given structure can ever be formed in some class of organisms. (Compare page 407.)

▪ **Undecidability in *Mathematica*.** In choosing functions to build into *Mathematica* I tried to avoid ones that would often encounter undecidability. And this is why for example there is no built-in function in *Mathematica* that tries to predict whether a given program will terminate. But inevitably functions like *FixedPoint*, *ReplaceRepeated* and *FullSimplify* can run into undecidability—so that ultimately they have to be limited by constructs such as *$IterationLimit* and *TimeConstraint*.

▪ **Undecidability and sets.** Functions that can be computed in finite time by systems like Turing machines are often called recursive (or effectively computable). Sets are called recursive if there is a recursive function that can test whether or not any given element is in them. Sets are called recursively enumerable if there is a recursive function that can eventually generate any element in them.

The set of initial conditions for which a given Turing machine halts is thus not recursive. But it turns out that this set is recursively enumerable. And the reason is that one can generate the elements in it by effectively maintaining a copy of the Turing machine for each possible initial condition, then following a procedure where for example at step n one updates the one for initial condition *IntegerExponent[n, 2]*, and watches to see if it halts. Note that while the complement of a recursive set is always recursive, the complement of a recursively enumerable set may not be recursively enumerable. (An example is the set of initial conditions for which a Turing machines does not halt.) Recursively enumerable sets are characteristically associated with so-called Σ_1 statements of the form $\exists_t \phi[t]$ (where ϕ is recursive). (Asking whether a system ever halts is equivalent to asking whether there exists a number of steps t at which the system can be determined to be in its halting state.) Complements of recursively enumerable sets are characteristically associated with Π_1 statements of the form $\forall_t \phi[t]$—an example being whether a given system never halts. (Π_1 and Σ_1 statements are such that if they can be shown to be undecidable, then respectively they must be true or false, as discussed on page 1167.) If a statement in minimal form involves n alternations of $\exists$ and $\forall$ it is Σ_{n+1} if it starts with $\exists$ and Π_{n+1} if it starts with $\forall$. The Π_n and Σ_n form the so-called arithmetic hierarchy in which statements with larger n can be constructed by allowing ϕ to access an oracle for statements with smaller n (see page 1126). (Showing that a statement with $n \geq 1$ is undecidable does not establish that it is always true or always false.)

▪ **Undecidability in tiling problems.** The question of whether a particular set of constraints like those on page 220 can be satisfied over the whole 2D plane is in general undecidable. For much as on page 943, one can imagine setting up a 1D cellular automaton with the property that, say, the absence of a particular color of cell throughout the 2D pattern formed by its evolution signifies satisfaction of the constraints. But even starting from a fixed line of cells, the question of whether a given color will ever occur in the evolution of a 1D cellular automaton is in general undecidable, as discussed in the main text. And although it is somewhat more difficult to show, this question remains undecidable even if one allows any possible configuration of cells on the starting line. (There are several different detailed formulations; the first explicit proof of undecidability in a tiling problem was given by Hao Wang in 1960; the version with no fixed cells by Robert Berger in 1966 by setting up an elaborate emulation of a register machine.) (See also page 943.)

▪ **Page 757 · Correspondence systems.** Given a list of pairs p with *{u, v} = Transpose[p]* the constraint to be satisfied is

StringJoin[u[[s]]] == StringJoin[v[[s]]]

Thus for example *p = {{"ABB", "B"}, {"B", "BA"}, {"A", "B"}}* has shortest solution *s = {2, 3, 2, 2, 3, 2, 1, 1}*. (One can have lists instead of strings, replacing *StringJoin* by *Flatten*.)

Correspondence systems were introduced by Emil Post in 1945 to give simple examples of undecidability; he showed that the so-called Post Correspondence Problem (PCP) of satisfying their constraints is in general undecidable (see below). With 2 string pairs PCP was shown to be decidable in 1981. It is known to be undecidable when 9 pairs are used, but I strongly suspect that it is also undecidable with just 3 pairs. The undecidability of PCP has been used to establish undecidability of many problems related to groups, context-free languages, and other objects defined by relations (see page 1141). Finding PCP solutions shorter than a given length is known to be an NP-complete problem.

With r string pairs and *n = StringLength[StringJoin[p]]* there are 2^n *Binomial[n – 1, 2 r – 1]* possible constraints (assuming no strings of zero length), each being related to at most $8\,r!$ others by straightforward symmetries (or altogether 4^{n-1} for given n). The number of constraints which yield solutions of specified lengths *Length[s]* for $r = 2$ and $r = 3$ are as follows (the boxes at the end give the number of cases with no solution):

r = 2, n = 4	*1:12*	*4*							
r = 2, n = 5	*1:64*	*64*							
r = 2, n = 6	*1:208*	*2:28*	*404*						
r = 2, n = 7	*1:640*	*3:32*	*1888*						
r = 2, n = 8	*1:1680*	*2:176*	*4:48*	*7056*					
r = 2, n = 9	*1:4352*	*3:112*	*5:56*	*24152*					
r = 2, n = 10	*1:10496*	*2:744*	*3:80*	*4:168*	*6:64*	*74464*			
r = 3, n = 6	*1:56*	*8*							
r = 3, n = 7	*1:576*	*192*							
r = 3, n = 8	*1:3312*	*2:168*	*3:84*	*1812*					
r = 3, n = 9	*1:14592*	*2:1140*	*3:192*	*4:288*	*5:96*	*8:48*	*30:48*	*44:48*	*12220*
r = 3, n = 10	*1:55296*	*2:4752*	*3:2712*	*4:372*	*5:492*	*6:264*	*7:216*		
	12:24	*18:48*	*24:48*	*36:48*	*75:48*	*78:48*	*64656*		

With $r = 2$, as n increases an exponentially decreasing fraction of possible constraints have solutions; with $r = 3$ it appears that a fraction more than 1/4 continue to do so. With $r = 2$, it appears that if a solution exists, it must have length $n + 4$ or less. With $r \leq 3$, the longest minimal solution lengths for $n \leq 10$ are given above. (Allowing $r > 3$ yields no greater lengths for these values of n.) With $n = 11$, example (l) yields a solution of length 112. The only possible longer $n = 11$ case is *{{"AAB", "B"}, {"B", "A"}, {"A", "AABB"}}*, for which

any possible solution must be longer than 200. With *n = 12*, *{{"AABAAB", "B"}, {"B", "A"}, {"A", "AB"}}* has minimal solution length 120 and *{{"A", "AABB"}, {"AAB", "B"}, {"B", "AA"}}* has minimal solution length 132.

A given constraint can fail to have a solution either because the colors of cells at some point cannot be made to match, or because the two strings can never have the same finite length (as in *{{"A", "AA"}}*). To know that a solution exists in a particular case, it is sufficient just to exhibit it. To know that no solution is possible of any length, one must in effect have a proof.

In general, one condition for a solution to exist is that integer numbers of pairs can yield strings of the same length, so that given the length differences *d = Map[StringLength, p, {2}] . {1, -1}* there is a vector *v* of non-negative integers such that *v . d == 0*. If only one color of element ever appears this is the complete condition for a solution—and for *r = 2* solutions exist if *Apply[Times, d] < 0* and are then of length at least *Apply[Plus[##]/GCD[##] &, Abs[d]]*. With two colors of elements additional conditions can be constructed involving counting elements of each color, or various blocks of elements.

The undecidability of PCP can be seen to follow from the undecidability of the halting problem through the fact that the question of whether a tag system of the kind on page 93 with initial sequence *s* ever reaches a halting state (where none of its rules apply) is equivalent to the question of whether there is a way to satisfy the PCP constraint

```
TSToPCP[{n_, rule_}, s_] :=
 Map[Flatten[IntegerDigits[#, 2, 2]] &, Module[{f}, f[u_] :=
    Flatten[Map[{1, #} &, 3 u]]; Join[Map[{f[Last[#]],
       RotateLeft[f[First[#]]]} &, rule], {{f[s], {1}}}, Flatten[
     Table[{{1, 2}, Append[RotateLeft[f[IntegerDigits[j, 2,
          i]]], 2]}, {i, 0, n - 1}, {j, 0, 2^i - 1}], 1]]], {2}]
```

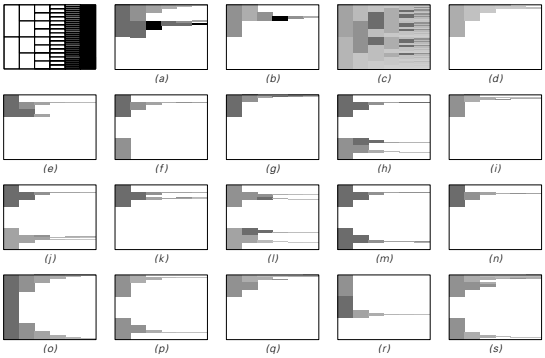
Any PCP constraint can also immediately be related to the evolution of a multiway tag system of the kind discussed in the note below. Assuming that the upper string is never shorter than the lower one, the rules for the relevant tag system are given simply by

```
Apply[Append[#2, s___] → Prepend[#1, s] &, p, {1}]
```

In the case of example (e) the existence of a solution of length 24 can then be seen to follow from the fact that *MWTSEvolve[rule, {{"B"}}, 22]* contains *{"B", "A"}*.

This correspondence with tag systems can be used in practice to search for PCP solutions, though it is usually most efficient to run tag systems that correspond both to moving forward and backward in the string, and to see whether their results ever agree. (In most PCP systems, including all the examples shown except (a) and (g), one string is always systematically longer than the other.) The tag system approach is normally limited by the number of intermediate strings that may need to be kept.

The pictures below show which possible sequences of up to 6 blocks yield upper and lower strings that agree in each of the PCP systems in the main text. As indicated in the first picture for the case of two blocks, each possible successively longer sequence corresponds to a rectangle in the picture (compare page 594). When a sequence of blocks leads to upper and lower strings that disagree, the rectangle is left white. If the strings agree so far, then the rectangle is colored with a gray that is darker if the strings are closer in length. Rectangles that are black (as visible in cases (a) and (b)) correspond to actual PCP solutions where the strings are the same length. Note that in case (c) the presence of only one color in either block means that strings will always agree so far. In cases (m) through (s) there is ultimately no solution, but as the pictures indicate, in these specific PCP systems there are always strings that agree as far as they have gone—it is just that they never end up the same length.



As one example of how one proves that a PCP constraint cannot be satisfied, consider case (s). From looking at the structure of the individual pairs one can see that if there is a solution it must begin with pair 1 or pair 3, and end with pair 1. But in fact it cannot begin with pair 1 because this would mean that the upper string would have to start off being longer, then at some point cross over to being shorter. However, the only way that such a crossover can occur is by pair 3 appearing with its upper *A* aligned with its second lower *A*. Yet starting with pair 1, the upper string is longer by 2 *A*s, and the pairs are such that the length difference must always remain even—preventing the crossover from occurring. This means that any solution must begin with pair 3. But this pair must then be followed by another pair 3, which leaves *BAAB* sticking out on the bottom. So how can

this *BAAB* be removed? The only way is to use the sequence of pairs 2, 3, 3, 2—yet doing this will just produce another *BAAB* further on. And thus one concludes that there is no way to satisfy these particular PCP constraints.

One can generalize PCP to allow any number of colors, and to require correspondence among any number of strings—though it is fairly easy to translate any such generalization to the 2-string 2-color case.

▪ **Multiway tag systems.** As an extension of ordinary multiway systems one can generalize tag systems from page 93 to allow a list of strings at each step. Representing the strings by lists, one can write rules in the form

```
{{1, 1, s___} → {s, 1, 0}, {1, s___} → {s, 1, 0, 1}}
```

so that the evolution is given by

```
MWTSEvolve[rule_, list_, t_] :=
  Nest[Flatten[Map[ReplaceList[#, rule] &, #], 1] &, list, t]
```

▪ **Word problems.** The question of whether a particular string can be generated in a given multiway system is an example of a so-called word problem. An original more specialized version of this was posed by Max Dehn in 1911 for groups and by Axel Thue in 1914 for semigroups. As discussed on page 938 a finitely presented group or semigroup can be viewed as a special case of a multiway system, in which the rules of the multiway system are obtained from relations between strings consisting of products of generators. The word problem then asks if a given product of such generators is equal to the identity element. Following work by Alan Turing in the mid-1930s, it was shown in 1947 by Emil Post from the undecidability of PCP that the word problem for semigroups is in general undecidable. Andrei Markov gave a specific example of this for a semigroup with 13 generators and 33 relations, and by 1966 Gennadiĭ Makanin had found the simpler example

```
{"CCBB" ↔ "BBCC", "BCCCBB" ↔ "CBBBCC", "ACCBB" ↔ "BBA",
  "ABCCCBB" ↔ "CBBA", "BBCCBBBBCC" ↔ "BBCCBBBBCCA"}
```

Using these relations as rules for a multiway system most initial strings yield behavior that either dies out or becomes repetitive. The shortest initial strings that give unbounded growth are *"BBBBABB"* and *"BBBBBBA"*—though both of these still eventually yield just exponentially increasing numbers of distinct strings. In 1967 Yuri Matiyasevich constructed a semigroup with 3 complicated relations that has an undecidable word problem. It is not yet known whether undecidability can occur in a semigroup with a single relation. The word problem is known to be decidable for commutative semigroups.

The word problem for groups was shown to be undecidable in the mid-1950s by Petr Novikov and William Boone. There are however various classes of groups for which it is decidable. Abelian groups are one example. Another are so-called automatic groups, studied particularly in the 1980s, in which equivalence of words can be recognized by a finite automaton. (Such groups turn out to have definite geometrical properties, and are associated with spaces of negative curvature.) Even if a group ultimately has only a finite number of distinct elements, its word problem (with elements specified as products of generators) may still be undecidable. Constructions of groups with undecidable word problems have been based on setting up relations that correspond to the rules in a universal Turing machine. With the simplest such machine known in the past (see page 706) one gets a group with 32 generators and 142 relations. But with the universal Turing machine from page 707 one gets a group with 14 generators and 52 relations. (In general $s\,k + 4$ generators and $5\,s\,k + 2$ relations are needed.) From the results in this book it seems likely that there are still much simpler examples—some of which could perhaps be found by setting up groups to emulate rule 110. Note that groups with just one relation were shown always to have decidable word problems by Wilhelm Magnus in 1932.

For ordinary multiway (semi-Thue) systems, an example with an undecidable word problem is known with 2 types of elements and 5 very complicated rules—but I am quite certain that much simpler examples are possible. (1-rule multiway systems always have decidable word problems.)

▪ **Sequence equations.** One can ask whether by replacing variables by sequences one can satisfy so-called word or string equations such as

```
Flatten[{x, 0, x, 0, y}] == Flatten[{y, x, 0, y, 1, 0, 1, 0, 0}]
```

(with shortest solution $x = \{1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0\}$, $y = \{1, 0, 1, 0, 0, 1, 0, 1, 0, 0\}$). Knowing about PCP and Diophantine equations one might expect that in general this would be undecidable. But in 1977 Gennadiĭ Makanin gave a complicated algorithm that solves the problem completely in a finite number of steps (though in general triple exponential in the length of the equation).

▪ **Fast algorithms.** Most of the fast algorithms now known seem to fall into a few general classes. The most common are ones based on repetition or iteration, classic examples being Euclid's algorithm for *GCD* (page 915), Newton's method for *FindRoot* and the Gaussian elimination method for *LinearSolve*. Starting in the 1960s it began to be realized that fast algorithms could be based on nested or recursive processes, and such algorithms became increasingly popular in the 1980s. In most cases, the idea is recursively to divide data into parts, then to do operations on these parts, and

finally reassemble the results. An example is the algorithm of Anatolii Karatsuba from 1961 for finding products of n-digit numbers (with $n = 2^s$) by operating on their digits in the nested pattern of page 608 (see also page 1093) according to

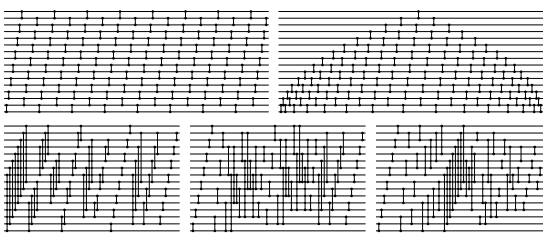
```
First[f[IntegerDigits[x, 2, n], IntegerDigits[y, 2, n], n/2]]

f[x_, y_, n_] :=
  If[n < 1, x y, g[Partition[x, n], Partition[y, n], n]]
g[{x1_, x0_}, {y1_, y0_}, n_] :=
  With[{z1 = f[x1, y1, n/2], z0 = f[x0, y0, n/2]},
   z1 2^(2 n) + (f[x0 + x1, y0 + y1, n/2] - z1 - z0) 2^n + z0]
```

Other examples include the fast Fourier transform (page 1074) and related algorithms for *ListConvolve*, the quicksort algorithm for *Sort*, and many algorithms in fields such as computational geometry. Starting in the 1980s fast algorithms based on randomized methods (see page 1192) have also become popular. But particularly from the discoveries in this book, it seems likely that the very fastest algorithms for many kinds of problems will not in the end have the type of regular structure that characterizes almost all algorithms currently used.

■ **Sorting networks.** Any list can be sorted using *Fold[PairSort, list, pairs]* by doing a fixed sequence of comparisons of pairs

```
PairSort[a_, p : {_, _}] := Block[{t = a}, t[[p]] = Sort[t[[p]]]; t]
```

(Different comparisons often do not interfere and so can be done in parallel.) The pictures below show a few sequences of pair comparisons that sort lists of length $n = 16$.



The top two (both with 120 comparisons) have a repetitive structure and correspond to standard sorting algorithms: transposition sort and insertion sort. (Quicksort does not use a fixed sequence of comparisons.) The first one on the bottom (with 63 comparisons) has a nested structure and uses the method invented by Kenneth Batcher in 1964:

```
Flatten[Reverse[Flatten[With[{m = Ceiling[Log[2, n]] - 1},
    Table[With[{d = If[i == m, 2^t, 2^(i+1) - 2^t]}, Map[
      {0, d} + # &, Select[Range[n - d], BitAnd[# - 1, 2^t] ==
        If[i == m, 0, 2^t] &]]], {t, 0, m}, {i, t, m}]], 1]], 1]
```

The second one on the bottom also uses 63 comparisons, while the last one is the smallest known for $n = 16$: it uses 60 comparisons and was invented by Milton Green in 1969. For $n \le 16$ the smallest numbers of comparisons known to work are $\{0, 1, 3, 5, 9, 12, 16, 19, 25, 29, 35, 39, 45, 51, 56, 60\}$. (In general all lists will be sorted correctly if lists of just 0's and 1's are sorted correctly; allowing even just one of these 2^n cases to be wrong greatly reduces the number of comparisons needed.) For $n \le 8$ the Batcher method is known to give minimal length sequences of comparisons (for $n \le 5$ the total numbers of minimal sequences that work are $\{1, 6, 3, 13866\}$). The Batcher method in general requires about $n\,Log[n]^2$ comparisons; it is known that in principle $n\,Log[n]$ are sufficient. Various structures such as de Bruijn and Cayley graphs can be used as the basis for sorting networks, though it is my guess that typically the smallest networks for given n will have no obvious regularity. (See also page 832.)

■ **Page 758 · Computational complexity theory.** Despite its rather general name, computational complexity theory has for the most part been concerned with the quite specific issue of characterizing how the computational resources needed to solve problems grow with input size. From knowing explicit algorithms many problems can be assigned to such classes as:

- NC: can be solved in a number of steps that increases like a polynomial in the logarithm of the input size if processing is done in parallel on a number of arbitrarily connected processors that increases like a polynomial in the input size. (Examples include addition and multiplication.)
- P (polynomial time): can be solved (with one processor) in a number of steps that increases like a polynomial in the input size. (Examples include evaluating standard mathematical functions and simulating the evolution of cellular automata and Turing machines.)
- NP (non-deterministic polynomial time): solutions can be checked in polynomial time. (Examples include many problems based on constraints as well as simulating the evolution of multiway systems and finding initial conditions that lead to given behavior in a cellular automaton.)
- PSPACE (polynomial space): can be solved with an amount of memory that increases like a polynomial in the input size. (Examples include finding repetition periods in systems of limited size.)

Central to computational complexity theory are a collection of hypotheses that imply that NC, P, NP and PSPACE form a strict hierarchy. At each level there are many problems known that are complete at that level in the sense that all other problems at that level can be translated to instances of that problem using only computations at a lower level. (Thus, for example, all problems in NP can be translated to instances of any given NP-complete problem using computations in P.)

■ **History.** Ideas of characterizing problems by growth rates in the computational resources needed to solve them were discussed in the 1950s, notably in the context of operation counts for numerical calculations, sizes of circuits for switching and other applications, and theoretical lengths of proofs. In the 1960s such ideas were increasingly formalized, particularly for execution times on Turing machines, and in 1965 the suggestion was made that one should consider computations feasible if they take times that grow like polynomials in their input size. NP completeness (see below) was introduced by Stephen Cook in 1971 and Leonid Levin around the same time. And over the course of the 1970s a great many well-known problems were shown to be NP-complete. A variety of additional classes of computations—notably ones like NC with various kinds of parallelism, ones based on circuits and ones based on algebraic operations—were defined in the 1970s and 1980s, and many detailed results about them were found. In the 1980s much work was also done on the average difficulty of solving NP-complete problems—both exactly and approximately (see page 985). When computational complexity theory was at its height in the early 1980s it was widely believed that if a problem could be shown, for example, to be NP-complete then there was little chance of being able to work with it in a practical situation. But increasingly it became clear that general asymptotic results are often quite irrelevant in typical problems of reasonable size. And certainly pattern matching with __ in *Mathematica*, as well as polynomial manipulation functions like *GroebnerBasis*, routinely deal with problems that are formally NP-complete.

■ **Lower bounds.** If one could prove for example that P ≠ NP then one would immediately have lower bounds on all NP-complete problems. But absent such a result most of the general lower bounds known so far are based on fairly straightforward information content arguments. One cannot for example sort n objects in less than about n steps since one must at least look at each object, and one cannot multiply two n-digit numbers in less than about n steps since one must at least look at each digit. (As it happens the fastest known algorithms for these problems require very close to n steps.) And if the output from a computation can be of size 2^n then this will normally take at least 2^n steps to generate. Subtleties in defining how big the input to a computation really is can lead to at least apparently exponential lower bounds. An example is testing whether one can match all possible sequences with a regular expression that involves s-fold repetitions. It is fairly clear that this cannot be done in less than about s steps. But this seems exponentially large if s is specified by its digit sequence in the original input regular expression. Similar issues arise in the problem of determining truth or falsity in Presburger arithmetic (see page 1152).

Diagonalization arguments analogous to those on pages 1128 and 1162 show that in principle there must exist functions that can be evaluated only by computations that exceed any given bound. But actually finding examples of such functions that can readily be described as having some useful purpose has in the past seemed essentially impossible.

If one sufficiently restricts the form of the underlying system then it sometimes becomes possible to establish meaningful lower bounds. For example, with deterministic finite automata (see page 957), there are sequences that can be recognized, but only by having exponentially many states. And with DNF Boolean expressions (see page 1096) functions like *Xor* are known to require exponentially many terms, even—as discovered in the 1980s—if any limited number of levels are allowed (see page 1096).

■ **Algorithmic complexity theory.** Ordinary computational complexity theory asks about the resources needed to run programs that perform a given computation. But algorithmic complexity theory (compare page 1067) asks instead about how large the programs themselves need to be. The results of this book indicate however that even programs that are very small—and thus have low algorithmic complexity—can nevertheless perform all sorts of complex computations.

■ **Turing machines.** The Turing machines used here in effect have tapes that extend only to the left, and have no explicit halt states. (They thus differ from the Turing machines which Marvin Minsky and Daniel Bobrow studied in 1961 in the $s = 2$, $k = 2$ case and concluded all had simple behavior.) One can think of each Turing machine as computing a function $f[x]$ of the number x given as its input. The function is total (i.e. defined for all x) if the Turing machine always halts; otherwise it is partial (and undefined for at least some x). Turing machines can be numbered according to the scheme on page 888. The number of steps before a machine with given rule halts can be computed from (see page 888)

```
Module[{s = 1, a, i = 1, d}, a[_] = 0; MapIndexed[a[#2[[1]]] =
  #1 &, Reverse[IntegerDigits[x, 2]]]; Do[{s, a[i], d} =
  {s, a[i]} /. rule; i -= d; If[i == 0, Return[t]], {t, tmax}]]
```

Of the 4096 Turing machines with $s = 2$, $k = 2$, 748 never halt, 3348 sometimes halt and 1683 always halt. (The most rarely halting are ones like machine 3112 that halt only when $x = 4 j - 1$.) The number of distinct functions $f[x]$ that can be computed by such machines is 351, of which 149 are total. 17 machines compute $x + 1$; none compute $x + 2$; 17 compute $x - 1$ and do not halt when $x = 0$—an example being 2575. Most machines compute functions that involve digit manipulations

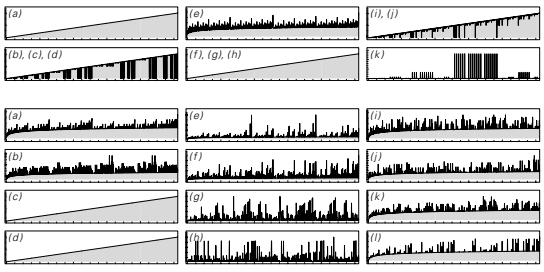
without traditional interpretations as mathematical functions. It is quite common to find machines that compute almost the same function: 1507 and 1511 disagree (where 1507 halts) only for $x \geq 35$. If $t[x]$ is the number of steps to compute $f[x]$ then the number of distinct pairs $\{f[x], t[x]\}$ is 492, or 230 for total $f[x]$. In 164 $t[x]$ does not increase with the number of digits n in x, in 295 it increases linearly, in 27 quadratically, and in 6 exponentially. For total $f[x]$ the corresponding numbers are 84, 136, 7, 3; the 3 machines with exponential growth are 378 (example (f) on page 761), 1953 and 2289; all compute trivial functions. Machine 1447 (example (e)) computes the function which takes the digit sequence of x and replaces its first $3 + \text{IntegerExponent}[x + 1, 2]$ 0's by 1's.

Among the 2,985,984 Turing machines with $s = 3$, $k = 2$, at least 2,550,972 sometimes halt, and about 1,271,304 always do. The number of distinct functions that can be computed is about 36,392 (or 75,726 for $\{f[x], t[x]\}$ pairs). 8934 machines compute $x + 1$ (by 25 different methods, including ones like machine 164850 that take exponential steps), 14 compute $x + 2$, and none compute $x + 3$. Those machines that take times that grow precisely like 2^n all tend to compute very straightforward functions which can be computed much faster by other machines.

Among the 2,985,984 Turing machines with $s = 2$, $k = 3$, at least 2,760,721 sometimes halt, and about 974,595 always halt. The number of distinct functions that can be computed is about 315,959 (or 457,508 for $\{f[x], t[x]\}$ pairs). (The fact that there are far fewer distinct functions in the $s = 3$, $k = 2$ case is a consequence of equivalences between states but not colors.)

Among the 2^{32} Turing machines with $s = 4$, $k = 2$ about 80% at least sometimes halt, and about 16% always do. Still none compute $x + 3$. And no Turing machine of any size can directly compute a function like x^2, $2x$ or $\text{Mod}[x, 2]$ that involves manipulating all digits in x.

■ **Functions.** The plots below show the values of the functions $f[x]$ for x from 0 to 1023 computed by the Turing machines on pages 761 and 763. Many of the plots use logarithmic scales. Rarely are the values close to their absolute maximum $t[x]$.



■ **Machine 1507.** This machine shows in some ways the most complicated behavior of any $s = 2$, $k = 2$ Turing machine. As suggested by picture (k) it fails to halt if and only if its configuration at some step matches $\{(0)..., \{1, 1\}, 1, \_\_\_\}$ (in the alternative form of page 888). For any input x one can test whether the machine will ever halt using

```
u[{Reverse[IntegerDigits[x, 2]], 0}]

u[list_] := v[Split[Flatten[list]]]

v[{a_, b_ : {}, c_ : {}, d_ : {}, e_ : {}, f_ : {}, g___}] :=
  Which[a == {1} || First[a] == 0, True, c == {}, False,
    EvenQ[Length[b]], u[{a, 1 - b, c, d, e, f, g}],
    EvenQ[Length[c]], u[{a, 1 - b, c, 1, Rest[d], e, f, g, 0}],
    e == {} || Length[d] ≥ Length[b] + Length[a] - 2,
    True, EvenQ[Length[e]], u[{a, b, c, d, f, g}],
    True, u[{a, 1 - b, c, 1 - d, e, 1, Rest[f], g, 0}]]
```

This test takes at most $n/3$ recursive steps, even though the original machine can take of order n^2 steps to halt. Among $s = 3$, $k = 2$ machines there are 314 machines that do the same computation as 1507, but none any faster.

■ **Page 763 · Properties.** The maximum numbers of steps increase with input size according to:

(a) $14\,2\,\text{^}\,\text{Floor}[n/2] - 11 + 2\,\text{Mod}[n, 2]$

(b) (does not halt for $x = 1$)

(c) $2^n - 1$

(d) $(7\,(1 + \text{Mod}[n, 2])\,4\,\text{^}\,\text{Floor}[n/2] + 2\,\text{Mod}[n, 2] - 7)/3$

(h) (see note below)

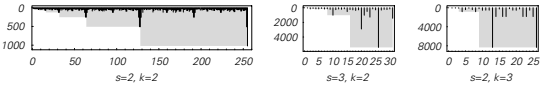
(i) (does not halt for various $x > 53$)

(j) (does not halt for various $x > 39$)

(k) (does not halt for $x = 1$)

(l) $5\,(2^{n-2} - 1)$

■ **Longest halting times.** The pictures below show the largest numbers of steps $t[x]$ that it takes any machine of a particular type to halt when given successive inputs x. For $s = 2$, $k = 2$ the largest results for all inputs of sizes 0 to 4 are $\{7, 17, 31, 49, 71\}$, all obtained with machine 1447. For $n > 4$ the largest results are $2^{n+2} - 3$, achieved for $x = 2^n - 1$ with machines 378 and 1351. For $s = 3$, $k = 2$ the largest results for successive sizes are $\{25, 53, 159, 179, 1021, 5419\}$ (often achieved by machine 600720; see below) and for $s = 2$, $k = 3$ $\{35, 83, 843, 8335\}$ (often achieved by machine 840971). Note the similarity to the busy beaver problem discussed on page 889.



▪ **Growth rates.** Some Turing machine can always be found that has halting times that grow at any specified rate. (See page 103 for a symbolic system with halting times that grow like `Nest[2^# &, 0, n]`.) As discussed on page 1162, if the growth rate is too high then it may not be possible to prove that the machines halt using, say, the standard axioms of arithmetic. The maximum halting times above increase faster than the halting times for any specific Turing machine, and are therefore ultimately not computable by any single Turing machine.

▪ **Machine 600720.** (Case (h) of page 763.) The maximum halting times for the first few sizes n are

```
{5, 159, 161, 1021, 5419, 315391,
  1978213883, 1978213885, 3018415453261}
```

These occur for inputs `{1, 2, 5, 10, 26, 34, 106, 213, 426}` and correspond to outputs (each themselves maximal for given n)

```
2^{3, 23, 24, 63, 148, 1148, 91148, 91149, 3560523} - 1
```

Such maxima often seem to occur when the input x has the form $(20\,4^s - 2)/3$ (and so has digits $\{1, 1, 0, 1, 0, \ldots, 1, 0\}$). The output `f[x]` in such cases is always $2^u - 1$ where

```
u = Nest[(13 + (6 # + 8) (5/2)^
                IntegerExponent[6 # + 8, 2])/6 &, 1, s + 1]
```

One then finds that $6\,u + 8$ has the form `Nest[If[EvenQ[#], 5#/2, # + 21] &, 14, m]` for some m, suggesting a connection with the number theory systems of page 122. The corresponding halting time `t[x]` is `Last[Nest[h, {8, 4 s + 24}, s]] - 1` with

```
h[{i_, j_}] := With[{e = IntegerExponent[3 i + 4, 2]}, {13/6 +
      (i + 4/3) (5/2)^(e+1), ((154 + 75 (i + 4/3) (5/2)^e)^2 -
        16321 - 7860 i - 900 i^2 + 3360 e)/3780 + j}]
```

For $s > 3$ it then turns out that `f[x]` is extremely close to $3560523\,(5/2)^r$, and `t[x]` to $188650989793 73\,(5/2)^{2r}$, for some integer r.

It is very difficult in general to find traditional formulas for `f[x]` and `t[x]`. But if `IntegerDigits[x, 2]` involves no consecutive 0's then for example `f[x]` can be obtained from

```
2^(b[Join[{1, 1}, #], Length[#]] &)[IntegerDigits[x, 2]] - 1

a[{l_, _}, r_] := ({l + (5 r - 3 #)/2, #} &)[Mod[r, 2]]

a[{l_, 0}, 0] := {l + 1, 0}

a[{l_, 1}, 0] :=
  ({(13 + # (5/2)^IntegerExponent[#, 2])/6, 0} &)[6 l + 2]

b[list_, i_] := First[Fold[a, {Apply[Plus, Drop[list, -i]], 0},
    Apply[Plus, Split[Take[list, -i], #1 == #2 ≠ 0 &], 1]]]
```

(The corresponding expression for `t[x]` is more complicated.) A few special cases are:

```
f[4 s] = 4 s + 3

f[4 s + 1] = 2 f[2 s] + 1

f[2^s - 1] = 2^((10 s+5+3 (-1)^s)/4 - 1
```

How the halting times behave for large n is not clear. It is certainly possible that they could increase like `NestList[#^2 &, 2, n]`, or 2^{2^n}, although for $x = (20\,4^s - 2)/3$ a better fit for $n \lesssim 200$ is just $2^{2.6\,n}$, with outputs increasing like $2^{2^{1.3\,n}}$.

▪ **Page 766 · NP completeness.** Among the hundreds of problems known to be NP-complete are:

- Can a non-deterministic Turing machine reach a certain state in a given number of steps?
- Can a multiway system generate a certain string in a given number of steps?
- Is there an assignment of truth values to variables that makes a given Boolean expression true? (Satisfiability; related to minimal Boolean expressions of page 1095.)
- Will a given sequence of pair comparisons correctly sort any list (see page 1142)?
- Will a given pattern of origami folds yield an object that can be made flat?
- Does a network have any parts that match a given subnetwork (see page 1038)?
- Is there a path shorter than some given length that visits all of some set of points in the plane? (Travelling salesman; related to the network layout problem of page 1031.)
- Is there a solution of a certain size to an integer linear programming problem?
- Is there any $x < a$ such that `Mod[x^2, b] == c`? (See page 1090.)
- Does a matrix have a permanent of given value?
- Is there a way to satisfy tiling constraints in a finite region? (See page 984.)
- Is there a string of some limited length that solves a correspondence problem?
- Is there an initial condition to a cellular automaton that yields particular behavior after a given number of steps?

(In cases where numbers are involved, it is usually crucial that these be represented by base 2 digit sequences, and not, say, in unary.) Many NP-complete problems at first seem quite unrelated. But often their equivalence becomes clear just by straightforward identification of terms. And so for example the equivalence of satisfiability to problems about networks can be seen by identifying variables and clauses in Boolean expressions respectively with connections and nodes in networks.

One can get an idea of the threshold of NP completeness by looking at seemingly similar problems where one is NP-complete but the other is in P. Examples include:

- Finding a Hamiltonian circuit that visits once every connection in a given network is NP-complete, but finding an Euler circuit that visits once every node is in P.
- Finding the longest path between two nodes in a network is NP-complete, but finding the shortest path is in P.
- Determining satisfiability for a Boolean expression with 3 variables in each clause is NP-complete, but for one with 2 variables is in P. (The latter is like a network with only 2 connections at each node.)
- Solving quadratic Diophantine equations $a x^2 + b y == c$ is NP-complete, but solving linear ones $a x + b y == c$ is in P.
- Finding a minimum energy configuration for a 2D Ising spin glass in a magnetic field is NP-complete, but is in P if there is no magnetic field.
- Finding the permanent of a matrix is NP-complete, but finding its determinant is in P.

It is not known whether problems such as integer factoring or equivalence of networks under relabelling of nodes (graph isomorphism) are NP-complete. It is known that in principle there exist NP problems that are not in P, yet are not NP-complete.

■ **Natural systems.** Finding minimum energy configurations is formally NP-complete in standard models of natural systems such as folding protein and DNA molecules (see page 1003), collections of charges on a sphere (compare page 987), and finite regions of spin glasses (see page 944). As discussed on page 351, however, it seems likely that in nature true minima are very rare, and that instead what is usually seen are just the results of actual dynamical processes of evolution.

In quantum field theory and to a lesser extent quantum mechanics and celestial mechanics, approximation schemes based on perturbation series seem to require computations that grow very rapidly with order. But exactly what this implies about the underlying physical processes is not clear.

■ **P versus NP questions.** Most programs that are explicitly constructed to solve specific problems tend at some level to have rather simple behavior—often just repetitive or nested, so long as appropriate number representations are used. And it is this that makes it realistic to estimate asymptotic growth rates using traditional mathematics, and to determine whether the programs operate in polynomial time. But as the pictures on page 761 suggest, arbitrary computational systems—even Turing machines with very simple rules—can exhibit much more complicated behavior with no clear asymptotic growth rate. And indeed the question of whether the halting times for a system grow only like a power of input size is in general undecidable. And if one tries to prove a result about halting times using, say, standard axioms of arithmetic or set theory, one may find that the result is independent of those axioms. So this makes it far from clear that the general P = NP question has a definite answer within standard axiom systems of mathematics. If one day someone were to find a provably polynomial time algorithm that solves an NP-complete problem then this would establish that P = NP. But it could well be that the fastest programs for NP-complete problems behave in ways that are too complicated to prove much about using the standard axioms of mathematics.

■ **Non-deterministic Turing machines.** Generalizing rules from page 888 by making each right-hand side a list of possible outcomes, the list of configurations that can be reached after t steps is given by

```
NTMEvolve[rule_, inits_, t_Integer] := Nest[
    Union[Flatten[Map[NTMStep[rule, #] &, #], 1]] &, inits, t]
NTMStep[rule_List, {s_, a_, n_}] /; 1 ≤ n ≤ Length[a] :=
  Apply[{#1, ReplacePart[a, #2, n], n + #3} &,
    Replace[{s, a[[n]]}, rule], {1}]
```

■ **Page 767 · Implementation.** Given a non-deterministic Turing machine with rules in the form above, the rules for a cellular automaton which emulates it can be obtained from

```
NDTMToCA[tm_] := Flatten[{{_, h, _} → h, {s, _c, _} → e, {s,
  _, _} → s, {_, s, c[i_]} → s[i], {_, s, x_} → x, {a[_, _], _s, _} → s,
  {_, a[x_, y_], s[i_]} → a[x, y, i], {x_, _s, _} → x, {_, _, s[i_]} →
  s[i], Map[Table[With[{b = (#[[Min[Length[#], z]]] &)[
    {x, #} /. tm]}, If[Last[b] == -1, {{a[_], a[x, #, z], e} → h, {a[
     _], a[x, #, z], s} → a[x, #, z], {a[_], a[x, #, z], _} → a[b[[2]]],
    {a[x, #, z], a[w_], _} → a[b[[1]], w], {_, a[w_], a[x, #, z]} →
    a[w]}, {{a[_], a[x, #, z], _} → a[b[[2]]], {a[x, #, z], a[w_],
    _} → a[w], {_, a[w_], a[x, #, z]} → a[b[[1]], w]}]], {x,
   Max[Map[#[[1, 1]] &, tm]]}, {z, Max[Map[Length[#[[2]]] &,
   tm]]}] &, Union[Map[#[[1, 2]] &, tm]]], {_, x_, _} → x}]
```

■ **Page 768 · Satisfiability.** Given variables $s[t, s]$, $a[t, x, a]$, $n[t, n]$ representing whether at step t a non-deterministic Turing machine is in state s, the tape square at position x has color a, and the head is at position n, the following CNF expression represents the assertion that a Turing machine with $stot$ states and $ktot$ possible colors follows the specified rules and halts after at most t steps:

```
NDTMToCNF[rules_, {s_, a_, n_}, t_] :=
 {Table[Apply[Or, Table[s[i, j], {j, stot}]], {i, t - 1}],
  Table[! s[i, j] || ! s[i, k], {i, 0, t - 1}, {j, stot}, {k, j + 1, stot}],
  Table[Apply[Or, Table[n[i, j], {j, n + i, Max[0, n - i], -2}]],
   {i, 0, t}], Table[! n[i, j] || ! n[i, k], {i, 0, t}, {j, n + i, Max[0,
    n - i], -2}, {k, j + 2, n + i}], Table[Apply[Or, Table[a[i, j, k],
    {k, 0, ktot - 1}]], {i, 0, t - 1}, {j, Max[1, n - i], n + i}],
  Table[! a[i, j, k] || ! a[i, j, m], {i, 0, t - 1}, {j, Max[1, n - i],
    n + i}, {k, 0, ktot - 1}, {m, k + 1, ktot - 1}], s[0, s],
```

```
Cases[MapIndexed[a[Abs[n - First[#2]], First[#2], #1] &,
 a], a[x_, _, _] /; x < t], Table[a[Abs[n - i], i, 0],
 {i, Length[a] + 1, n + t - 1}], Table[! a[i, j, k] ||
  If[EvenQ[n + i - j], n[i, j], False] || a[i + 1, j, k], {i, 0, t - 2},
 {j, Max[1, n - i], n + i}, {k, 0, ktot - 1}], Table[Map[Function[
  z, Outer[! n[i, j] || ! s[i, z[[1, 1]]] || ! a[i, j, z[[1, 2]]] || ## &,
   Apply[Sequence, Map[If[i < t - 1, {s[i + 1, #[[1]]], n[
       i + 1, j - #[[3]]], a[i + 1, j, #[[2]]]}, {n[i + 1, j - #[[3]]]}] &,
    z[[2]]]]]], rules], {i, 0, t - 1}, {j, n + i, Max[1, n - i], -2}],
 Apply[Or, Table[n[i, 0], {i, n, t, 2}]]] /. List → And
```

■ **Density of difficult problems.** There are arguments that in an asymptotic sense most instances chosen at random of problems like limited-size PCP or tiling will be difficult to solve. In a problem like satisfiability, however, difficult instances tend to occur only on the boundary between cases where the density of black or white squares implies that there is usually satisfaction or usually not satisfaction. If one looks at simple instances of problems (say PCP with short strings) then my experience is that many are easy to solve. But just as some fraction of cellular automata with very simple rules show immensely complex behavior, so similarly it seems that some fraction of even simple instances of many NP-complete problems also tend to be difficult to solve.

■ **Page 770 · Rule 30 inversion.** The total numbers of sequences for t from 1 to 15 not yielding stripes of heights 1 and 2 are respectively

```
{1, 2, 2, 3, 3, 6, 6, 10, 16, 31, 52, 99, 165, 260}
```

```
{2, 5, 8, 14, 23, 40, 66, 111, 182,
  316, 540, 921, 1530, 2543, 4122}
```

The sideways evolution of rule 30 discussed on page 601 implies that if one fills cells from the left rather than the right then some sequence of length $t + 1$ will always yield any given stripe of height t.

If the evolution of rule 30 can be set up as on page 704 to emulate any Boolean function then the problem considered here is immediately equivalent to satisfiability.

■ **Systems of limited size.** In the system $x \to Mod[x + m, n]$ from page 255 the repetition period $n/GCD[m, n]$ can be computed using Euclid's algorithm in at most about $Log[GoldenRatio, n]$ steps. In the system $x \to Mod[2\,x, n]$ from page 257, the repetition period $MultiplicativeOrder[2, n]$ probably cannot always be computed in any polynomial of $Log[n]$ steps, since otherwise $FactorInteger[n]$ could also be computed in about this number of steps. (But see note below.) In a cellular automaton with n cells, the problem of finding the repetition period is in general PSPACE-complete—as follows from the possibility of universality in the underlying cellular automaton. And even in a case like rule 30 I suspect that the period cannot be found much faster than by tracing nearly 2^n steps of evolution. (I know of no way for example to break the computation into parts that can be done in parallel.) With sufficiently simple behavior, a cellular automaton repetition period can readily be determined in some power of $Log[n]$ steps. But even with an additive rule and nested behavior, the period depends on quantities like $MultiplicativeOrder[2, n]$, which probably take more like n steps to evaluate. (But see note below.)

■ **Page 771 · Quantum computers.** In an ordinary classical setup one typically describes the state of something like a 2-color cellular automaton with n cells just by giving a list of n color values. But the standard formalism of quantum theory (see page 1058) implies that for an analogous quantum system—like a line of n quantum spins each either up or down—one instead has to give a whole vector of probability amplitudes for each of the 2^n possible complete underlying spin configurations. And these amplitudes a_i are assumed to be complex numbers with a continuous range of possible values, subject only to the conventional constraint of unit total probability $Sum[Abs[a_i]^2, \{i, 2^n\}] == 1$. The evolution of such a quantum system can then formally be represented by successive multiplication of the vector of amplitudes by appropriate $2^n \times 2^n$ unitary matrices.

In a classical system like a cellular automaton with n cells a probabilistic ensemble of states can similarly be described by a vector of 2^n probabilities p_i—now satisfying $Sum[p_i, \{i, 2^n\}] == 1$, and evolving by multiplication with $2^n \times 2^n$ matrices having a single 1 in each row. (If the system is reversible—as in the quantum case—then the matrices are invertible.) But even if one assumes that all 2^n states in the ensemble somehow manage to evolve in parallel, it is still fairly clear that to do reliable computations takes essentially as much effort as evolving single instances of the underlying system. For even though the vector of probabilities can formally give outcomes for 2^n different initial conditions, any specific individual outcome could have probability as small as 2^{-n}—and so would take 2^n trials on average to detect.

The idea of setting up quantum analogs of systems like Turing machines and cellular automata began to be pursued in the early 1980s by a number of people, including myself. At first it was not clear what idealizations to make, but by the late 1980s—especially through the work of David Deutsch—the concept had emerged that a quantum computer should be described in terms of a network of basic quantum gates. The idea was to have say n quantum spins (each representing a so-called qubit), then to do computations much like in the reversible logic systems of page 1097 or the sorting networks of page 1142 by applying some appropriate sequence of elementary operations. It was found to be sufficient to do operations on just one and two spins at a time, and in fact it

was shown that any $2^n \times 2^n$ unitary matrix can be approximated arbitrarily closely by a suitable sequence of for example underlying 2-spin $\{x, y\} \to \{x, Mod[x + y, 2]\}$ operations (assuming values 0 and 1), together with 1-spin arbitrary phase change operations. Such phase changes can be produced by repeatedly applying a single irrational rotation, and using the fact that $Mod[h\, s, 2\, \pi]$ will eventually for some s come close to any given phase (see page 903). From the involvement of continuous numbers, one might at first imagine that it should be possible to do fundamentally more computations than can be done say in ordinary discrete cellular automata. But all the evidence is that—just as discussed on page 1128—this will not in fact be possible if one makes the assumption that at some level discrete must be used to set up the initial values of probability amplitudes.

From the fact that the basic evolution of an n-spin quantum system in effect involves superpositions of 2^n spin configurations one might however still imagine that in finite computations exponential speedups should be possible. And as a potential example, consider setting up a quantum computer that evaluates a given Boolean function—with its initial configurations of spins encoding possible inputs to the function, and the final configuration of a particular spin representing the output from the function. One might imagine that with such a computer it would be easy to solve the NP-complete problem of satisfiability from page 768: one would just start off with a superposition in which all 2^n possible inputs have equal amplitude, then look at whether the spin representing the output from the function has any amplitude to be in a particular configuration. But in an actual physical system one does not expect to be able to find values of amplitudes directly. For according to the standard formalism of quantum theory all amplitudes do is to determine probabilities for particular outcomes of measurements. And with the setup described, even if a particular function is ultimately satisfiable the probability for a single output spin to be measured say as up can be as little as 2^{-n}—requiring on average 2^n trials to distinguish from 0, just as in the classical probabilistic case.

With a more elaborate setup, however, it appears sometimes to be possible to spread out quantum amplitudes so as to make different outcomes correspond to much larger probability differences. And indeed in 1994 Peter Shor found a way to do this so as to get quantum computers at least formally to factor integers of size n using resources only polynomial in n. As mentioned in the note above, it becomes straightforward to factor m if one can get the values of *MultiplicativeOrder[a, m]*. But these correspond to periodicities in the list *Mod[a^Range[m], m]*. Given n spins one can imagine using their 2^n possible configurations to represent each element of *Range[m]*. But now if one sets up a superposition of all these configurations, one can compute $Mod[a^{\#}, m]\,\&$, then essentially use *Fourier* to find periodicities—all with a polynomial number of quantum gates. And depending on *FactorInteger[m]* the resulting amplitudes show fairly large differences which can then be detected in the probabilities for different outcomes of measurements.

In the mid-1990s it was thought that quantum computers might perhaps give polynomial solutions to all NP problems. But in fact only a very few other examples were found—all ultimately based on very much the same ideas as factoring. And indeed it now seems decreasingly likely that quantum computers will give polynomial solutions to NP-complete problems. (Factoring is not known to be NP-complete.)

And even in the case of factoring there are questions about the idealizations used. It does appear that only modest precision is needed for the initial amplitudes. And it seems that perturbations from the environment can be overcome using versions of error-correcting codes. But it remains unclear just what might be needed actually to perform for example the final measurements required.

Simple physical versions of individual quantum gates have been built using particles localized for example in ion traps. But even modestly larger setups have been possible only in NMR and optical systems—which show formal similarities to quantum systems (and for example exhibit interference) but presumably do not have any unique quantum advantage. (There are other approaches to quantum computation that involve for example topology of 4D quantum fields. But it is difficult to see just what idealizations are realistic for these.)

■ **Circuit complexity.** Any function with a fixed size of input can be computed by a circuit of the kind shown on page 619. How the minimal size or depth of circuit needed grows with input size then gives a measure of the difficulty of the computation, with circuit depth growing roughly like number of steps for a Turing machine. Note that much as on page 662 one can construct universal circuits that can be arranged by appropriate choice of parts of their input to compute any function of a given input size. (Compare page 703.)

■ **Page 771 · Finding outcomes.** If one sets up a function to compute the outcome after t steps of evolution from some fixed initial condition—say a single black cell in a cellular automaton—then the input to this function need contain only *Log[2, t]* digits. But if the evolution is computationally irreducible then to find its outcome will involve explicitly following each of its t steps—thereby effectively finding results for each of the *2^Log[2, t]* possible arrangements of

digits corresponding to numbers less than t. Note that the computation that is involved is not necessarily in either NP or PSPACE.

▪ **P completeness.** If one allows arbitrary initial conditions in a cellular automaton with nearest-neighbor rules, then to compute the color of a particular cell after t steps in general requires specifying as input the colors of all $2t+1$ initial cells up to distance t away (see page 960). And if one always does computations using systems that have only nearest-neighbor rules then just combining $2t+1$ bits of information can take up to t steps—even if the bits are combined in a way that is not computationally irreducible. So to avoid this one can consider systems that are more like circuits in which any element can get data from any other. And given t elements operating in parallel one can consider the class NC studied by Nicholas Pippenger in 1978 of computations that can be done in a number of steps that is at most some power of *Log[t]*. Among such computations are *Plus*, *Times*, *Divide*, *Det* and *LinearSolve* for integers, as well as determining outcomes in additive cellular automata (see page 609). But I strongly suspect that computational irreducibility prevents outcomes in systems like rule 30 and rule 110 from being found by computations that are in NC—implying in effect that allowing arbitrary connections does not help much in computing the evolution of such systems. There is no way yet known to establish this for certain, but just as with NP and P one can consider showing that a computation is P-complete with respect to transformations in NC. It turns out that finding the outcome of evolution in any standard universal Turing machine or cellular automaton is P-complete in this sense, since the process of emulating any such system by any other one is in NC. Results from the mid-1970s established that finding the output from an arbitrary circuit with *And* or *Or* gates is P-complete, and this has made it possible to show that finding the outcome of evolution in various systems not yet known to be universal is P-complete. A notable example due to Cristopher Moore from 1996 is the 3D majority cellular automaton with rule *UnitStep[a + AxesTotal[a, 3] - 4]* (see page 927); another example is the Ising model cellular automaton from page 982.

Implications for Mathematics and Its Foundations

▪ **History.** Babylonian and Egyptian mathematics emphasized arithmetic and the idea of explicit calculation. But Greek mathematics tended to focus on geometry, and increasingly relied on getting results by formal deduction. For being unable to draw geometrical figures with infinite accuracy this seemed the only way to establish anything with certainty. And when Euclid around 330 BC did his work on geometry he started from 10 axioms (5 "common notions" and 5 "postulates") and derived 465 theorems. Euclid's work was widely studied for more than two millennia and viewed as a quintessential example of deductive thinking. But in arithmetic and algebra—which in effect dealt mostly with discrete entities—a largely calculational approach was still used. In the 1600s and 1700s, however, the development of calculus and notions of continuous functions made use of more deductive methods. Often the basic concepts were somewhat vague, and by the mid-1800s, as mathematics became more elaborate and abstract, it became clear that to get systematically correct results a more rigid formal structure would be needed.

The introduction of non-Euclidean geometry in the 1820s, followed by various forms of abstract algebra in the mid-1800s, and transfinite numbers in the 1880s, indicated that mathematics could be done with abstract structures that had no obvious connection to everyday intuition. Set theory and predicate logic were proposed as ultimate foundations for all of mathematics (see note below). But at the very end of the 1800s paradoxes were discovered in these approaches. And there followed an increasing effort—notably by David Hilbert—to show that everything in mathematics could consistently be derived just by starting from axioms and then using formal processes of proof.

Gödel's Theorem showed in 1931 that at some level this approach was flawed. But by the 1930s pure mathematics had already firmly defined itself to be based on the notion of doing proofs—and indeed for the most part continues to do so even today (see page 859). In recent years, however, the increasing use of explicit computation has made proof less important, at least in most applications of mathematics.

▪ **Models of mathematics.** Gottfried Leibniz's notion in the late 1600s of a "universal language" in which arguments in mathematics and elsewhere could be checked with logic can be viewed as an early idealization of mathematics. Starting in 1879 with his "formula language" (*Begriffsschrift*) Gottlob Frege followed a somewhat similar direction, suggesting that arithmetic and from there all of mathematics could be built up from predicate logic, and later an analog of set theory. In the 1890s Giuseppe Peano in his *Formulario* project organized a large body of mathematics into an axiomatic framework involving logic and set theory. Then starting in 1910 Alfred Whitehead and Bertrand Russell in their *Principia Mathematica* attempted to derive many areas of mathematics from foundations of logic and set theory. And although its methods were flawed and its notation obscure this work did

much to establish the idea that mathematics could be built up in a uniform way.

Starting in the late 1800s, particularly with the work of Gottlob Frege and David Hilbert, there was increasing interest in so-called metamathematics, and in trying to treat mathematical proofs like other objects in mathematics. This led in the 1920s and 1930s to the introduction of various idealizations for mathematics—notably recursive functions, combinators, lambda calculus, string rewriting systems and Turing machines. All of these were ultimately shown to be universal (see page 784) and thus in a sense capable of reproducing any mathematical system. String rewriting systems—as studied particularly by Emil Post—are close to the multiway systems that I use in this section (see page 938).

Largely independent of mathematical logic the success of abstract algebra led by the end of the 1800s to the notion that any mathematical system could be represented in algebraic terms—much as in the operator systems of this section. Alfred Whitehead to some extent captured this in his 1898 *Universal Algebra*, but it was not until the 1930s that the theory of structures emphasized commonality in the axioms for different fields of mathematics—an idea taken further in the 1940s by category theory (and later by topos theory). And following the work of the Bourbaki group beginning at the end of the 1930s it has become almost universally accepted that structures together with set theory are the appropriate framework for all of pure mathematics.

But in fact the *Mathematica* language released in 1988 is now finally a serious alternative. For while it emphasizes calculation rather than proof its symbolic expressions and transformation rules provide an extremely general way to represent mathematical objects and operations—as for example the notes to this book illustrate.

(See also page 1176.)

■ **Page 773 · Axiom systems.** In the main text I argue that there are many consequences of axiom systems that are quite independent of their details. But in giving the specific axiom systems that have been used in traditional mathematics one needs to take account of all sorts of fairly complicated details.

As indicated by the tabs in the picture, there is a hierarchy to axiom systems in traditional mathematics, with those for basic and predicate logic, for example, being included in all others. (Contrary to usual belief my results strongly suggest however that the presence of logic is not in fact essential to many overall properties of axiom systems.)

As discussed in the main text (see also page 1155) one can think of axioms as giving rules for transforming symbolic expressions—much like rules in *Mathematica*. And at a fundamental level all that matters for such transformations is the structure of expressions. So notation like $a+b$ and $a \times b$, while convenient for interpretation, could equally well be replaced by more generic forms such as f[a, b] or g[a, b] without affecting any of the actual operation of the axioms.

My presentation of axiom systems generally follows the conventions of standard mathematical literature. But by making various details explicit I have been able to put all axiom systems in forms that can be used almost directly in *Mathematica*. Several steps are still necessary though to get the actual rules corresponding to each axiom system. First, the definitions at the top of page 774 must be used to expand out various pieces of notation. In basic logic I use the notation $u = v$ to stand for the pair of rules $u \to v$ and $v \to u$. (Note that $=$ has the precedence of $\to$ not $==$.) In predicate logic the tab at the top specifies how to construct rules (which in this case are often called rules of inference, as discussed on page 1155). $x\_ \wedge y\_ \to x\_$ is the *modus ponens* or detachment rule (see page 1155). $x\_ \to \forall_{y\_}\, x\_$ is the generalization rule. $x\_ \to x\_ \wedge \#\,\&$ is applied to the axioms given to get a list of rules. Note that while $=$ in basic logic is used in the underlying construction of rules, $==$ in predicate logic is just an abstract operator with properties defined by the last two axioms given.

As is typical in mathematical logic, there are some subtleties associated with variables. In the axioms of basic logic literal variables like a must be replaced with patterns like $a\_$ that can stand for any expression. A rule like $a\_ \wedge (b\_ \vee \neg b\_) \to a$ can then immediately be applied to part of an expression using Replace. But to apply a rule like $a\_ \to a \wedge (b \vee \neg b)$ requires in effect choosing some new expression for b (see page 1155). And one way to represent this process is just to have the pattern $a\_ \to a\_ \wedge (b\_ \vee \neg b\_)$ and then to say that any actual rule that can be used must match this pattern. The rules given in the tab for predicate logic work the same way. Note, however, that in predicate logic the expressions that appear on each side of any rule are required to be so-called well-formed formulas (WFFs) consisting of variables (such as a) and constants (such as 0 or $\emptyset$) inside any number of layers of functions (such as $+$, $\cdot$, or Δ) inside a layer of predicates (such as $==$ or $\in$) inside any number of layers of logical connectives (such as $\wedge$ or $\Rightarrow$) or quantifiers (such as $\forall$ or $\exists$). (This setup is reflected in the grammar of the *Mathematica* language, where the operator precedences for functions are higher than for predicates, which are in turn higher than for quantifiers and logical connectives—thus

yielding for example few parentheses in the presentation of axiom systems here.)

In basic logic any rule can be applied to any part of any expression. But in predicate logic rules can be applied only to whole expressions, always in effect using *Replace[expr, rules]*. The axioms below (devised by Matthew Szudzik as part of the development of this book) set up basic logic in this way.

$(a \vee b) \vee c = (b \vee a) \vee c$	$a \vee ((b \wedge c) \wedge d) = a \vee ((c \wedge b) \wedge d)$
$((a \vee b) \vee c) \vee d = (a \vee (b \vee c)) \vee d$	$a \vee (((b \wedge c) \wedge d) \wedge e) = a \vee ((b \wedge (c \wedge d)) \wedge e)$
$a \vee (b \wedge (c \wedge \neg c)) = a$	$a \vee (b \wedge (c \vee \neg c)) = a \vee b$
$a \vee (b \vee (c \wedge d)) = a \vee ((b \vee c) \wedge (b \vee d))$	$a \vee (b \wedge (c \vee d)) = a \vee ((b \wedge c) \vee (b \wedge d))$
$a \vee (b \wedge \neg (c \vee d)) = a \vee (b \wedge (\neg c \wedge \neg d))$	$a \vee (b \wedge \neg (c \wedge d)) = a \vee (b \wedge (\neg c \vee \neg d))$
$a \vee (b \wedge c) = a \vee (b \wedge \neg \neg c)$	

■ **Basic logic.** The formal study of logic began in antiquity (see page 1099), with verbal descriptions of many templates for valid arguments—corresponding to theorems of logic—being widely known by medieval times. Following ideas of abstract algebra from the early 1800s, the work of George Boole around 1847 introduced the notion of representing logic in a purely symbolic and algebraic way. (Related notions had been considered by Gottfried Leibniz in the 1680s.) Boole identified *1* with *True* and *0* with *False*, then noted that theorems in logic could be stated as equations in which *Or* is roughly *Plus* and *And* is *Times*—and that such equations can be manipulated by algebraic means. Boole's work was progressively clarified and simplified, notably by Ernst Schröder, and by around 1900, explicit axiom systems for Boolean algebra were being given. Often they included most of the 14 highlighted theorems of page 817, but slight simplifications led for example to the "standard version" of page 773. (Note that the duality between *And* and *Or* is no longer explicit here.) The "Huntington version" of page 773 was given by Edward Huntington in 1933, along with

$$\{(\neg \neg a) = a,\ (a \vee \neg (b \vee \neg b)) = a,\ (\neg (\neg (a \vee \neg b) \vee \neg (a \vee \neg c))) = (a \vee \neg (b \vee c))\}$$

The "Robbins version" was suggested by Herbert Robbins shortly thereafter, but only finally proved correct in 1996 by William McCune using automated theorem proving (see page 1157). The "Sheffer version" based on *Nand* (see page 1173) was given by Henry Sheffer in 1913. The shorter version was devised by David Hillman as part of the development of this book. The shortest version is discussed on page 808. (See also page 1175.)

In the main text each axiom defines an equivalence between expressions. The tradition in philosophy and mathematical logic has more been to take axioms to be true statements from which others can be deduced by the *modus ponens* inference rule $\{x, x \Rightarrow y\} \to y$ (see page 1155). In 1879 Gottlob Frege used his diagrammatic notation to set up a symbolic representation for logic on the basis of the axioms

$$\{a \Rightarrow (b \Rightarrow a),\ (a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c)),\ (a \Rightarrow (b \Rightarrow c)) \Rightarrow (b \Rightarrow (a \Rightarrow c)),\ (a \Rightarrow b) \Rightarrow ((\neg b) \Rightarrow (\neg a)),\ (\neg \neg a) \Rightarrow a,\ a \Rightarrow (\neg \neg a)\}$$

Charles Peirce did something similar at almost the same time, and by 1900 this approach to so-called propositional or sentential calculus was well established. (Alfred Whitehead and Bertrand Russell used an axiom system based on *Or* and *Not* in their original 1910 edition of *Principia Mathematica*.) In 1948 Jan Łukasiewicz found the single axiom version

$$\{((a \Rightarrow (b \Rightarrow a)) \Rightarrow ((((\neg c) \Rightarrow (d \Rightarrow (\neg e))) \Rightarrow ((c \Rightarrow (d \Rightarrow f)) \Rightarrow ((e \Rightarrow d) \Rightarrow (e \Rightarrow f)))) \Rightarrow g)) \Rightarrow (h \Rightarrow g)\}$$

equivalent for example to

$$\{((\neg a) \Rightarrow (b \Rightarrow (\neg c))) \Rightarrow ((a \Rightarrow (b \Rightarrow d)) \Rightarrow ((c \Rightarrow b) \Rightarrow (c \Rightarrow d))),\ a \Rightarrow (b \Rightarrow a)\}$$

It turns out to be possible to convert any axiom system that works with *modus ponens* (and supports the properties of $\Rightarrow$) into a so-called equational one that works with equivalences between expressions by using

```
Module[{a}, Join[Thread[axioms == a ⇒ a],
    {((a ⇒ a) ⇒ b) == b, ((a ⇒ b) ⇒ b) == (b ⇒ a) ⇒ a}]]
```

An analog of *modus ponens* for *Nand* is $\{x, x \barwedge (y \barwedge z)\} \to z$, and with this Jean Nicod found in 1917 the single axiom

$$\{(a \barwedge (b \barwedge c)) \barwedge ((e \barwedge (e \barwedge e)) \barwedge ((d \barwedge b) \barwedge ((a \barwedge d) \barwedge (a \barwedge d))))\}$$

which was highlighted in the 1925 edition of *Principia Mathematica*. In 1931 Mordechaj Wajsberg found the slightly simpler

$$\{(a \barwedge (b \barwedge c)) \barwedge (((d \barwedge c) \barwedge ((a \barwedge d) \barwedge (a \barwedge d))) \barwedge (a \barwedge (a \barwedge b)))\}$$

Such an axiom system can be converted to an equational one using

```
Module[{a}, With[{t = a ⊼ (a ⊼ a), i = #1 ⊼ (#2 ⊼ #2) &},
    Join[Thread[axioms == t], {i[t ⊼ (b ⊼ c), c] == t,
        i[t, b] == b, i[i[a, b], b] == i[i[b, a], a]}]]]
```

but then involves 4 axioms.

The question of whether any particular statement in basic logic is true or false is always formally decidable, although in general it is NP-complete (see page 768).

■ **Predicate logic.** Basic logic in effect concerns itself with whole statements (or "propositions") that are each either *True* or *False*. Predicate logic on the other hand takes into account how such statements are built up from other constructs—like those in mathematics. A simple statement in predicate logic is $\forall_x (\forall_y x == y) \vee \forall_x (\exists_y (\neg x == y))$, where $\forall$ is "for all" and $\exists$ is "there exists" (defined in terms of $\forall$ on page 774)—and this particular statement can be proved *True* from the axioms. In general statements in predicate logic can contain arbitrary so-called predicates, say *p[x]* or *r[x, y]*, that are each either *True* or *False* for given *x* and *y*. When predicate logic is used

as part of other axiom systems, there are typically axioms which define properties of the predicates. (In real algebra, for example, the predicate $>$ satisfies $a > b \Rightarrow a \neq b$.) But in pure predicate logic the predicates are not assumed to have any particular properties.

Notions of quantifiers like $\forall$ and $\exists$ were already discussed in antiquity, particularly in the context of syllogisms. The first explicit formulation of predicate logic was given by Gottlob Frege in 1879, and by the 1920s predicate logic had become widely accepted as a basis for mathematical axiom systems. (Predicate logic has sometimes also been used as a model for general reasoning—and particularly in the 1980s was the basis for several initiatives in artificial intelligence. But for the most part it has turned out to be too rigid to capture directly typical everyday reasoning processes.)

Monadic pure predicate logic—in which predicates always take only a single argument—reduces in effect to basic logic and is not universal. But as soon as there is even one arbitrary predicate with two arguments the system becomes universal (see page 784). And indeed this is the case even if one considers only statements with quantifiers $\forall\ \exists\ \forall$. (The system is also universal with one two-argument function or two one-argument functions.)

In basic logic any statement that is true for all possible assignments of truth values to variables can always be proved from the axioms of basic logic. In 1930 Kurt Gödel showed a similar result for pure predicate logic: that any statement that is true for all possible explicit values of variables and all possible forms of predicates can always be proved from the axioms of predicate logic. (This is often called Gödel's Completeness Theorem, but is not related to completeness of the kind I discuss on page 782 and elsewhere in this section.)

In discussions of predicate logic there is often much said about scoping of variables. A typical issue is that in, say, $\forall_x (\exists_y (\neg x == y))$, x and y are dummy variables whose specific names are not supposed to be significant; yet the names become significant if, say, x is replaced by y. In *Mathematica* most such issues are handled automatically. The axioms for predicate logic given here follow the work of Alfred Tarski in 1962 and use properties of $==$ to minimize issues of variable scoping.

(See also higher-order logics on page 1167.)

■ **Arithmetic.** Most of the Peano axioms are straightforward statements of elementary facts about arithmetic. The last axiom is a schema (see page 1156) that states the principle of mathematical induction: that if a statement is valid for $a = 0$, and its validity for $a = b$ implies its validity for $a = b + 1$, then it follows that the statement must be valid for all a. Induction was to some extent already used in antiquity—for example in Euclid's proof that there are always larger primes. It began to be used in more generality in the 1600s. In effect it expresses the idea that the integers form a single ordered sequence, and it provides a basis for the notion of recursion.

In the early history of mathematics arithmetic with integers did not seem to need formal axioms, for facts like $x + y == y + x$ appeared to be self-evident. But in 1861 Hermann Grassmann showed that such facts could be deduced from more basic ones about successors and induction. And in 1891 Giuseppe Peano gave essentially the Peano axioms listed here (they were also given slightly less formally by Richard Dedekind in 1888)—which have been used unchanged ever since. (Note that in second-order logic—and effectively set theory— $+$ and $\times$ can be defined just in terms of $\triangle$; see page 1160. In addition, as noted by Julia Robinson in 1948 it is possible to remove explicit mention of $+$ even in the ordinary Peano axioms, using the fact that if $c == a + b$ then $(\triangle a \times c) \times (\triangle b \times c) == \triangle(c \times c) \times (\triangle a \times b)$. Axioms 3, 4 and 6 can then be replaced by $a \times b == b \times a$, $a \times (b \times c) == (a \times b) \times c$ and $(\triangle a) \times (\triangle a \times b) == \triangle a \times (\triangle b \times (\triangle a))$. See also page 1163.)

The proof of Gödel's Theorem in 1931 (see page 1158) demonstrated the universality of the Peano axioms. It was shown by Raphael Robinson in 1950 that universality is also achieved by the Robinson axioms for reduced arithmetic (usually called Q) in which induction—which cannot be reduced to a finite set of ordinary axioms (see page 1156)—is replaced by a single weaker axiom. Statements like $x + y == y + x$ can no longer be proved in the resulting system (see pages 800 and 1169).

If any single one of the axioms given for reduced arithmetic is removed, universality is lost. It is not clear however exactly what minimal set of axioms is needed, for example, for the existence of solutions to integer equations to be undecidable (see page 787). (It is known, however, that essentially nothing is lost even from full Peano arithmetic if for example one drops axioms of logic such as $\neg\ \neg\ a == a$.)

A form of arithmetic in which one allows induction but removes multiplication was considered by Mojzesz Presburger in 1929. It is not universal, although it makes statements of size n potentially take as many as about 2^{2^n} steps to prove (though see page 1143).

The Peano axioms for arithmetic seem sufficient to support most of the whole field of number theory. But if as I believe there are fairly simple results that are unprovable from these axioms it may in fact be necessary to extend the Peano

axioms to make certain kinds of progress even in practical number theory. (See also page 1166.)

■ **Algebraic axioms.** Axioms like $a \circ (b \circ c) = (a \circ b) \circ c$ can be used in at least three ways. First, as equations which can be manipulated—like the axioms of basic logic—to establish whether expressions are equal. Second, as on page 773, as statements to be added to the axioms of predicate logic to yield results that hold for every possible system described by the axioms (say every possible semigroup). And third, as definitions of sets whose properties can be studied—and compared—using set theory. High-school algebra typically treats axioms as equations. More advanced algebra often uses predicate logic, but implicitly uses set theory whenever it addresses for example mappings between objects. Note that as discussed on page 1159 how one uses algebraic axioms can affect issues of universality and undecidability. (See also page 1169.)

■ **Groups.** Groups have been used implicitly in the context of geometrical symmetries since antiquity. In the late 1700s specific groups began to be studied explicitly, mainly in the context of permutations of roots of polynomials, and notably by Evariste Galois in 1831. General groups were defined by Arthur Cayley around 1850 and their standard axioms became established by the end of the 1800s. The alternate axioms given in the main text are the shortest known. The first for ordinary groups was found by Graham Higman and Bernhard Neumann in 1952; the second by William McCune (using automated theorem proving) in 1992. For commutative (Abelian) groups the first alternate axioms were found by Alfred Tarski in 1938; the second by William McCune (using automated theorem proving) in 1992. In this case it is known that no shorter axioms are possible. (See page 806.) Note that in terms of the $\bar{\circ}$ operator $1 == a \bar{\circ} a$, $\overline{b} == (a \bar{\circ} a) \bar{\circ} b$, and $a \cdot b == a \bar{\circ} ((a \bar{\circ} a) \bar{\circ} b)$. Ordinary group theory is universal; commutative group theory is not (see page 1159).

■ **Semigroups.** Despite their simpler definition, semigroups have been much less studied than groups, and there have for example been about 7 times fewer mathematical publications about them (and another 7 times fewer about monoids). Semigroups were defined by Jean-Armand de Séguier in 1904, and beginning in the late 1920s a variety of algebraic results about them were found. Since the 1940s they have showed up sporadically in various areas of mathematics—notably in connection with evolution processes, finite automata and category theory.

■ **Fields.** With $\oplus$ being $+$ and $\otimes$ being $\times$ rational, real and complex numbers are all examples of fields. Ordinary integers lack inverses under $\times$, but reduction modulo a prime p gives a finite field. Since the 1700s many examples of fields have arisen, particularly in algebra and number theory. The general axioms for fields as given here emerged around the end of the 1800s. Shorter versions can undoubtedly be found. (See page 1168.)

■ **Rings.** The axioms given are for commutative rings. With $\oplus$ being $+$ and $\otimes$ being $\times$ the integers are an example. Several examples of rings arose in the 1800s in number theory and algebraic geometry. The study of rings as general algebraic structures became popular in the 1920s. (Note that from the axioms of ring theory one can only expect to prove results that hold for any ring; to get most results in number theory, for example, one needs to use the axioms of arithmetic, which are intended to be specific to ordinary integers.) For non-commutative rings the last axiom given is replaced by $(a \oplus b) \otimes c == a \otimes c \oplus b \otimes c$. Non-commutative rings already studied in the 1800s include quaternions and square matrices.

■ **Other algebraic systems.** Of algebraic systems studied in traditional mathematics the vast majority are special cases of either groups, rings or fields. Probably the most common other examples are those based on lattice theory. Standard axioms for lattice theory are ($\wedge$ is usually called meet, and $\vee$ join)

$$\{a \wedge b == b \wedge a,\ a \vee b == b \vee a,\ (a \wedge b) \wedge c == a \wedge (b \wedge c),\ (a \vee b) \vee c == a \vee (b \vee c),\ a \wedge (a \vee b) == a,\ a \vee a \wedge b == a\}$$

Boolean algebra (basic logic) is a special case of lattice theory, as is the theory of partially ordered sets (of which the causal networks in Chapter 9 are an example). The shortest single axiom currently known for lattice theory has *LeafCount* 79 and involves 7 variables. But I suspect that in fact a *LeafCount* less than about 20 is enough.

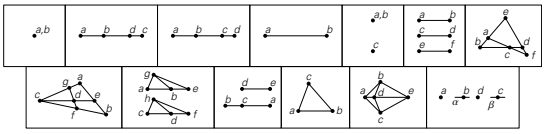
(See also page 1171.)

■ **Real algebra.** A notion of real numbers as measures of space or quantity has existed since antiquity. The development of basic algebra gave a formal way to represent operations on such numbers. In the late 1800s there were efforts—notably by Richard Dedekind and Georg Cantor—to set up a general theory of real numbers relying only on basic concepts about integers—and these efforts led to set theory. For purely algebraic questions of the kind that might arise in high-school algebra, however, one can use just the axioms given here. These add to field theory several axioms for ordering, as well as the axiom at the bottom expressing a basic form of continuity (specifically that any polynomial which changes sign must have a zero). With these axioms one can prove results about real polynomials, but not about arbitrary

mathematical functions, or integers. The axioms were shown to be complete by Alfred Tarski in the 1930s. The proof was based on setting up a procedure that could in principle resolve any set of real polynomial equations or inequalities. This is now in practice done by *Simplify* and other functions in *Mathematica* using methods of cylindrical algebraic decomposition invented in the 1970s—which work roughly by finding a succession of points of change using *Resultant*. (Note that with n variables the number of steps needed can increase like 2^{2^n}.) (See the note about real analysis below.)

▪ **Geometry.** Euclid gave axioms for basic geometry around 300 BC which were used with fairly little modification for more than 2000 years. In the 1830s, however, it was realized that the system would remain consistent even if the so-called parallel postulate was modified to allow space to be curved. Noting the vagueness of Euclid's original axioms there was then increasing interest in setting up more formal axiom systems for geometry. The best-known system was given by David Hilbert in 1899—and by describing geometrical figures using algebraic equations he showed that it was as consistent as the underlying axioms for numbers.

The axioms given here are illustrated below. They were developed by Alfred Tarski and others in the 1940s and 1950s. (Unlike Hilbert's axioms they require only first-order predicate logic.) The first six give basic properties of betweenness of points and congruence of line segments. The second- and third-to-last axioms specify that space has two dimensions; they can be modified for other dimensions. The last axiom is a schema that asserts the continuity of space. (The system is not finitely axiomatizable.)



The axioms given can prove most of the results in an elementary geometry textbook—indeed all results that are about geometrical figures such as triangles and circles specified by a fixed finite number of points, but which do not involve concepts like area. The axioms are complete and consistent—and thus not universal. They can however be made universal if axioms from set theory are added.

▪ **Category theory.** Developed in the 1940s as a way to organize constructs in algebraic topology, category theory works at the level of whole mathematical objects rather than their elements. In the basic axioms given here the variables represent morphisms that correspond to mappings between objects. (Often morphisms are shown as arrows in diagrams, and objects as nodes.) The axioms specify that when morphisms are composed their domains and codomains must have appropriately matching types. Some of the methodology of category theory has become widely used in mathematics, but until recently the basic theory itself was not extensively studied—and its axiomatic status remains unclear. Category theory can be viewed as a formalization of operations on abstract data types in computer languages—though unlike in *Mathematica* it normally requires that functions take a single bundle of data as an argument.

▪ **Set theory.** Basic notions of finite set theory have been used since antiquity—though became widespread only after their introduction into elementary mathematics education in the 1960s. Detailed ideas about infinite sets emerged in the 1880s through the work of Georg Cantor, who found it useful in studying trigonometric series to define sets of transfinite numbers of points. Several paradoxes associated with infinite sets were quickly noted—a 1901 example due to Bertrand Russell being to ask whether a set containing all sets that do not contain themselves in fact contains itself. To avoid such paradoxes Ernst Zermelo in 1908 suggested formalizing set theory using the first seven axioms given in the main text. (The axiom of infinity, for example, was included to establish that an infinite set such as the integers exists.) In 1922 Abraham Fraenkel noted that Zermelo's axioms did not support certain operations that seemed appropriate in a theory of sets, leading to the addition of Thoralf Skolem's axiom of replacement, and to what is usually called Zermelo-Fraenkel set theory (ZF). (The replacement axiom formally makes the subset axiom redundant.) The axiom of choice was first explicitly formulated by Zermelo in 1904 to capture the idea that in a set all elements can be ordered, so that the process of transfinite induction is possible (see page 1160). The non-constructive character of the axiom of choice has made it always remain somewhat controversial. It has arisen in many different guises and been useful in proving theorems in many areas of mathematics, but it has seemingly peculiar consequences such as the Banach-Tarski result that a solid sphere can be divided into six pieces (each a non-measurable set) that can be reassembled into a solid sphere twice the size. (The nine axioms with the axiom of choice are usually known as ZFC.) The axiom of regularity (or axiom of foundation) formulated by John von Neumann in 1929 explicitly forbids sets which for example can be elements of themselves. But while this axiom is convenient in simplifying work in set theory it has not been found generally useful in mathematics, and is normally considered optional at best.

A few additional axioms have also arisen as potentially useful. Most notable is the Continuum Hypothesis discussed on page 1127, which was proved independent of ZFC by Paul Cohen in 1963. (See also page 1166.)

Note that by using more complicated axioms the only construct beyond predicate logic needed to formulate set theory is $\in$. As discussed on page 1176, however, one cannot avoid axiom schemas in the formulation of set theory given here. (The von Neumann-Bernays-Gödel formulation does avoid these, but at the cost of introducing additional objects more general than sets.)

(See also page 1160.)

▪ **General topology.** The axioms given define properties of open sets of points in spaces—and in effect allow issues like connectivity and continuity to be discussed in terms of set theory without introducing any explicit distance function.

▪ **Real analysis.** The axiom given is Dedekind's axiom of continuity, which expresses the connectedness of the set of real numbers. Together with set theory it allows standard results about calculus to be derived. But as well as ordinary real numbers, these axioms allow non-standard analysis with constructs such as explicit infinitesimals (see page 1172).

▪ **Axiom systems for programs.** (See pages 794 and 1168.)

▪ **Page 775 · Implementation.** Given the axioms in the form

```
s[1] = (a_ ⊼ a_) ⊼ (a_ ⊼ b_) → a;
s[2, x_] := b_ → (b ⊼ b) ⊼ (b ⊼ x); s[3] =
  a_ ⊼ (a_ ⊼ b_) → a ⊼ (b ⊼ b); s[4] = a_ ⊼ (b_ ⊼ b_) → a ⊼ (a ⊼ b);
s[5] = a_ ⊼ (a_ ⊼ (b_ ⊼ c_)) → b ⊼ (b ⊼ (a ⊼ c));
```

the proof shown here can be represented by

```
{{s[2, b], {2}}, {s[4], {}}, {s[2, (b ⊼ b) ⊼ ((a ⊼ a) ⊼ (b ⊼ b))],
    {2, 2}}, {s[1], {2, 2, 1}}, {s[2, b ⊼ b], {2, 2, 2, 2, 2, 2}},
  {s[5], {2, 2, 2}}, {s[2, b ⊼ b], {2, 2, 2, 2, 2, 1}},
  {s[1], {2, 2, 2, 2, 2}}, {s[3], {2, 2, 2}},
  {s[1], {2, 2, 2, 2}}, {s[4], {2, 2, 2}}, {s[5], {}},
  {s[2, a], {2, 2, 1}}, {s[1], {2, 2}}, {s[3], {}}, {s[1], {2}}}
```

and applied using

```
FoldList[Function[{u, v},
    MapAt[Replace[#, v[[1]] &, u, {v[[2]]}]], a ⊼ b, proof]
```

▪ **Page 776 · Proof structures.** The proof shown is in a sense based on very low-level steps, each consisting of applying a single axiom from the original axiom system. But in practical mathematics it is usual for proofs to be built up in a more hierarchical fashion using intermediate results or lemmas. In the way I set things up lemmas can in effect be introduced as new axioms which can be applied repeatedly during a proof. And in the case shown here if one first proves the lemma

```
(a ⊼ (a ⊼ (b ⊼ ((a ⊼ a) ⊼ c)))) = (b ⊼ a)
```

and treats it as rule 6, then the main proof can be shortened:

When one just applies axioms from the original axiom system one is in effect following a single line of steps. But when one proves a lemma one is in effect on a separate branch, which only merges with the main proof when one uses the lemma. And if one has nested lemmas one can end up with a proof that is in effect like a tree. (Repeated use of a single lemma can also lead to cycles.) Allowing lemmas can in extreme cases probably make proofs as much as exponentially shorter. (Note that lemmas can also be used in multiway systems.)

In the way I have set things up one always gets from one step in a proof to the next by taking an expression and applying some transformation rule to it. But while this is familiar from algebraic mathematics and from the operation of *Mathematica* it is not the model of proofs that has traditionally been used in mainstream mathematical logic. For there one tends to think not so much about transforming expressions as about taking collections of true statements (such as equations $u == v$), and using so-called rules of inference to deduce other ones. Most often there are two basic rules of inference: *modus ponens* or detachment which uses the logic result $(x \wedge x \Rightarrow y) \Rightarrow y$ to deduce the statement y from statements x and $x \Rightarrow y$, and substitution, which takes statements x and y and deduces $x /. p \to y$, where p is a logical variable in x (see page 1151). And with this approach axioms enter merely as initial true statements, leaving rules of inference to generate successive steps in proofs. And instead of being mainly linear sequences of results, proofs instead become networks in which pairs of results are always combined when *modus ponens* is used. But it is still always in principle possible to convert any proof to a purely sequential one—though perhaps at the cost of having exponentially many more steps.

■ **Substitution strategies.** With the setup I am using each step in a proof involves transforming an expression like $u = v$ using an expression like $s = t$. And for this to happen s or t must match some part w of u or v. The simplest way this can be achieved is for s or t to reproduce w when its variables are replaced by appropriate expressions. But in general one can make replacements not only for variables in s and t, but also for ones in w. And in practice this often makes many more matches possible. Thus for example the axiom $a \circ a = a$ cannot be applied directly to $(p \circ q) \circ (p \circ r) = q \circ r$. But after the replacement $r \to q$, $a \circ a$ matches $(p \circ q) \circ (p \circ r)$ with $a \to p \circ q$, yielding the new theorem $p \circ q = q \circ q$. These kinds of substitutions are used in the proof on page 810. One approach to finding them is so-called paramodulation, which was introduced around 1970 in the context of automated theorem-proving systems, and has been used in many such systems (see page 1157). (Such substitutions are not directly relevant to *Mathematica*, since it transforms expressions rather than theorems or equations. But when I built SMP in 1981, its semantic pattern matching mechanism did use essentially such substitutions.)

■ **One-way transformations.** As formulated in the main text, axioms define two-way transformations. One can also set up axiom systems based on one-way transformations (as in multiway systems). For basic logic, examples of this were studied in the mid-1900s, and with the transformations thought of as rules of inference they were sometimes known as "axiomless formulations".

■ **Axiom schemas.** An axiom like `a + 0 == a` is a single well-formed formula in the sense of page 1150. But sometimes one needs infinite collections of such individual axioms, and in the main text these are represented by axiom schemas given as *Mathematica* patterns involving objects like `x_`. Such schemas are taken to stand for all individual axioms that match the patterns and are well-formed formulas. The induction axiom in arithmetic is an example of a schema. (See the note on finite axiomatizability on page 1176.) Note that as mentioned on page 1150 all the axioms given for basic logic should really be thought of as schemas.

■ **Reducing axiom details.** Traditional axiom systems have many details not seen in the basic structure of multiway systems. But in most cases these details can be avoided—and in the end the universality of multiway systems implies that they can always be made to emulate any axiom system.

Traditional axiom systems tend to be based on operator systems (see page 801) involving general expressions, not just strings. But any expression can always be written as a string using something like *Mathematica* `FullForm`. (See also page 1169.) Traditional axiom systems also involve symbolic variables, not just literal string elements. But by using methods like those for combinators on page 1121 explicit mention of variables can always be eliminated.

■ **Proofs in practice.** At some level the purpose of a proof is to establish that something is true. But in the practice of modern mathematics proofs have taken on a broader role; indeed they have become the primary framework for the vast majority of mathematical thinking and discourse. And from this perspective the kinds of proofs given on pages 810 and 811—or typically generated by automated theorem proving—are quite unsatisfactory. For while they make it easy at a formal level to check that certain statements are true, they do little at a more conceptual level to illuminate why this might be so. And indeed the kinds of proofs normally considered most mathematically valuable are ones that get built up in terms of concepts and constructs that are somehow expected to be as generally applicable as possible. But such proofs are inevitably difficult to study in a uniform and systematic way (though see page 1176). And as I argue in the main text, it is in fact only for the rather limited kinds of mathematics that have historically been pursued that such proofs can be expected to be sufficient. For in general proofs can be arbitrarily long, and can be quite devoid of what might be considered meaningful structure.

Among practical proofs that show signs of this (and whose mathematical value is thus often considered controversial) most have been done with aid of computers. Examples include the Four-Color Theorem (coloring of maps), the optimality of the Kepler packing (see page 986), the completeness of the Robbins axiom system (see page 1151) and the universality of rule 110 (see page 678).

In the past it was sometimes claimed that using computers is somehow fundamentally incompatible with developing mathematical understanding. But particularly as the use of *Mathematica* has become more widespread there has been increasing recognition that computers can provide crucial raw material for mathematical intuition—a point made rather forcefully by the discoveries in this book. Less well recognized is the fact that formulating mathematical ideas in a *Mathematica* program is at least as effective a way to produce clarity of thinking and understanding as formulating a traditional proof.

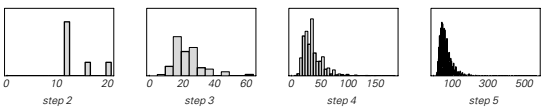
■ **Page 778 · Properties.** The second rule shown has the property that black elements always appear before white, so that strings can be specified just by the number of elements of each color that they contain—making the rule one of the sorted type discussed on page 937, based on the difference

vector *{{2, -1}, {-1, 3}, {-4, -1}}*. The question of whether a given string can be generated is then analogous to finding whether there is a solution with certain positivity properties to a set of linear Diophantine equations.

▪ **Page 781 · NAND tautologies.** At each step every possible transformation rule in the axioms is applied wherever it can. New expressions are also created by replacing each possible variable with $x \barwedge y$, where x and y are new variables, and by setting every possible pair of variables equal in turn. The longest tautology at step t is

Nest[(# ⊼ #) ⊼ (# ⊼ p_t) &, p ⊼ (p ⊼ p), t - 1]

whose *LeafCount* grows like 3^t. The distribution of sizes of statements generated at each step is shown below.



Even with the same underlying axioms the tautologies are generated in a somewhat different order if one uses a different strategy—say one based on paramodulation (see page 1156). Pages 818 and 1175 discuss the sequence of all NAND theorems listed in order of increasing complexity.

▪ **Proof searching.** To find a proof of some statement $p = q$ in a multiway system one can always in principle just start from p, evolve the system until it first generates q, then pick out the sequence of strings on the path from p to q. But doing this will usually involve building up a vast network of strings. And although at some level computational irreducibility and NP completeness (see page 766) imply that in general only a limited amount of this computational work can be saved, there are in practice quite often important optimizations that can be made. For finding a proof of $p = q$ is like searching for a path satisfying the constraint of going from p to q. And just like in the systems based on constraints in Chapter 5 one can usually do at least somewhat better than just to look at every possible path in turn.

For a start, in generating the network of paths one only ever need keep a single path that leads to any particular string; just like in many of my pictures of multiway systems one can in effect always drop any duplicate strings that occur. One might at first imagine that if p and q are both short strings then one could also drop any very long strings that are produced. But as we have seen, it is perfectly possible for long intermediate strings to be needed to get from p to q. Still, it is often reasonable to weight things so that at least at first one looks at paths that involve only shorter strings.

In the most direct approach, one takes a string and at each step just applies the underlying rules or axioms of the multiway system. But as soon as one knows that there is a path from a string u to a string v, one can also imagine applying the rule $u \to v$ to any string—in effect like a lemma. And one can choose which lemmas to try first by looking for example at which involve the shortest or commonest strings.

It is often important to minimize the number of lemmas one has to keep. Sometimes one can do this by reducing every lemma—and possibly every string—to some at least partially canonical form. One can also use the fact that in a multiway system if $u \to v$ and $r \to s$ then $u <> r \to v <> s$.

If one wants to get from p to q the most efficient thing is to use properties of q to avoid taking wrong turns. But except in systems with rather simple structure this is usually difficult to achieve. Nevertheless, one can for example always in effect work forwards from p, and backwards from q, seeing whether there is any overlap in the sets of strings one gets.

▪ **Automated theorem proving.** Since the 1950s a fair amount of work has been done on trying to set up computer systems that can prove theorems automatically. But unlike systems such as *Mathematica* that emphasize explicit computation none of these efforts have ever achieved widespread success in mathematics. And indeed given my ideas in this section this now seems not particularly surprising.

The first attempt at a general system for automated theorem proving was the 1956 Logic Theory Machine of Allen Newell and Herbert Simon—a program which tried to find proofs in basic logic by applying chains of possible axioms. But while the system was successful with a few simple theorems the searches it had to do rapidly became far too slow. And as the field of artificial intelligence developed over the next few years it became widely believed that what would be needed was a general system for imitating heuristics used in human thinking. Some work was nevertheless still done on applying results in mathematical logic to speed up the search process. And in 1963 Alan Robinson suggested the idea of resolution theorem proving, in which one constructs $\neg$ *theorem* $\vee$ *axioms*, then typically writes this in conjunctive normal form and repeatedly applies rules like $(\neg p \vee q) \wedge (p \vee q) \to q$ to try to reduce it to *False*, thereby proving given *axioms* that *theorem* is *True*. But after early enthusiasm it became clear that this approach could not be expected to make theorem proving easy—a point emphasized by the discovery of NP completeness in the early 1970s. Nevertheless, the approach was used with some success, particularly in proving that various mechanical and other engineering systems would behave as intended—although by the mid-1980s such verification was more often done by systematic Boolean

function methods (see page 1097). In the 1970s simple versions of the resolution method were incorporated into logic programming languages such as Prolog, but little in the way of mathematical theorem proving was done with them. A notable system under development since the 1970s is the Boyer-Moore theorem prover Nqthm, which uses resolution together with methods related to induction to try to find proofs of statements in a version of LISP. Another family of systems under development at Argonne National Laboratory since the 1960s are intended to find proofs in pure operator (equational) systems (predicate logic with equations). Typical of this effort was the Otter system started in the mid-1980s, which uses the resolution method, together with a variety of ad hoc strategies that are mostly versions of the general ones for multiway systems in the previous note. The development of so-called unfailing completion algorithms (see page 1037) in the late 1980s made possible much more systematic automated theorem provers for pure operator systems—with a notable example being the Waldmeister system developed around 1996 by Arnim Buch and Thomas Hillenbrand.

Ever since the 1970s I at various times investigated using automated theorem-proving systems. But it always seemed that extensive human input—typically from the creators of the system—was needed to make such systems actually find non-trivial proofs. In the late 1990s, however, I decided to try the latest systems and was surprised to find that some of them could routinely produce proofs hundreds of steps long with little or no guidance. Almost any proof that was easy to do by hand almost always seemed to come out automatically in just a few steps. And the overall ability to do proofs—at least in pure operator systems—seemed vastly to exceed that of any human. But as page 810 illustrates, long proofs produced in this way tend to be difficult to read—in large part because they lack the higher-level constructs that are typical in proofs created by humans. As I discuss on page 821, such lack of structure is in some respects inevitable. But at least for specific kinds of theorems in specific areas of mathematics it seems likely that more accessible proofs can be created if each step is allowed to involve sophisticated computations, say as done by *Mathematica*.

▪ **Proofs in *Mathematica*.** Most of the individual built-in functions of *Mathematica* I designed to be as predictable as possible—applying transformations in definite ways and using algorithms that are never of fundamentally unknown difficulty. But as their names suggest *Simplify* and *FullSimplify* were intended to be less predictable—and just to do what they can and then return a result. And in many cases these functions end up trying to prove theorems; so for example *FullSimplify[(a + b)/2 ≥ Sqrt[a b], a > 0 && b > 0]* must in effect prove a theorem to get the result *True*.

▪ **Page 781 · Truth and falsity.** The notion that statements can always be classified as either true or false has been a common idealization in logic since antiquity. But in everyday language, computer languages and mathematics there are many ways in which this idealization can fail. An example is $x + y == z$, which cannot reasonably be considered either true or false unless one knows what x, y and z are. Predicate logic avoids this particular kind of case by implicitly assuming that what is meant is a general statement about all values of any variable—and avoids cases like the expression $x + y$ by requiring all statements to be well-formed formulas (see page 1150). In *Mathematica* functions like *TrueQ* and *IntegerQ* are set up always to yield *True* or *False*—but just by looking at the explicit structure of a symbolic expression.

Note that although the notion of negation seems fairly straightforward in everyday language it can be difficult to implement in computational or mathematical settings. And thus for example even though it may be possible to establish by a finite computation that a particular system halts, it will often be impossible to do the same for the negation of this statement. The same basic issue arises in the intuitionistic approach to mathematics, in which one assumes that any object one handles must be found by a finite construction. And in such cases one can set up an analog of logic in which one no longer takes $\neg\ \neg a = a$.

It is also possible to assume a specific number $k > 2$ of truth values, as on page 1175, or to use so-called modal logics.

(See also page 1167.)

▪ **Page 782 · Gödel's Theorem.** What is normally known as "Gödel's Theorem" (or "Gödel's First Incompleteness Theorem") is the centerpiece of the paper "On Undecidable Propositions of *Principia Mathematica* and Related Systems" published by Kurt Gödel in 1931. What the theorem shows is that there are statements that can be formulated within the standard axiom system for arithmetic but which cannot be proved true or false within that system. Gödel's paper does this first for the statement "this statement is unprovable", and much of the paper is concerned with showing how such a statement can be encoded within arithmetic. Gödel in effect does this by first converting the statement to one about recursive functions and then—by using tricks of number theory such as the beta function of page 1120—to one purely about arithmetic. (Gödel's main achievement is sometimes characterized as the "arithmetization of metamathematics": the discovery that concepts such as provability related to the

processes of mathematics can be represented purely as statements in arithmetic.) (See page 784.)

Gödel originally based his theorem on Peano arithmetic (as discussed in the context of *Principia Mathematica*), but expected that it would in fact apply to any reasonable formal system for mathematics—and in later years considered that this had been established by thinking about Turing machines. He suggested that his results could be avoided if some form of transfinite hierarchy of formalisms could be used, and appears to have thought that at some level humans and mathematics do this (compare page 1167).

Gödel's 1931 paper came as a great surprise, although the issues it addressed were already widely discussed in the field of mathematical logic. And while the paper is at a technical level rather clear, it has never been easy for typical mathematicians to read. Beginning in the late 1950s its results began to be widely known outside of mathematics, and by the late 1970s Gödel's Theorem and various misstatements of it were often assigned an almost mystical significance. Self-reference was commonly seen as its central feature, and connections with universality and computation were usually missed. And with the belief that humans must somehow have intrinsic access to all truths in mathematics, Gödel's Theorem has been used to argue for example that computers can fundamentally never emulate human thinking.

The picture on page 786 can be viewed as a modern proof of Gödel's Theorem based on Diophantine equations.

In addition to what is usually called Gödel's Theorem, Kurt Gödel established a second incompleteness theorem: that the statement that the axioms of arithmetic are consistent cannot be proved by using those axioms (see page 1168). He also established what is often called the Completeness Theorem for predicate logic (see page 1152)—though here "completeness" is used in a different sense.

▪ **Page 783 · Properties.** The first multiway system here generates all strings that end in □■; the third all strings that end in ■. The second system generates all strings where the second-to-last element is white, or the string ends with a run of black elements delimited by white ones.

▪ **Page 783 · Essential incompleteness.** If a consistent axiom system is complete this means that any statement in the system can be proved true or false using its axioms, and the question of whether a statement is true can always be decided by a finite procedure. If an axiom system is incomplete then this means that there are statements that cannot be proved true or false using its axioms—and which must therefore be considered independent of those axioms. But even given this it is still possible that a finite procedure can exist which decides whether a given statement is true, and indeed this happens in the theory of commutative groups (see note below). But often an axiom system will not only be incomplete, but will also be what is called essentially incomplete. And what this means is that there is no finite set of axioms that can consistently be added to make the system complete. A consequence of this is that there can be no finite procedure that always decides whether a given statement is true—making the system what is known as essentially undecidable. (When I use the term "undecidable" I normally mean "essentially undecidable". Early work on mathematical logic sometimes referred to statements that are independent as being undecidable.)

One might think that adding rules to a system could never reduce its computational sophistication. And this is correct if with suitable input one can always avoid the new rules. But often these rules will allow transformations that in effect short-circuit any sophisticated computation. And in the context of axiom systems, adding axioms can be thought of as putting more constraints on a system—thus potentially in effect forcing it to be simpler. The result of all this is that an axiom system that is universal can stop being universal when more axioms are added to it. And indeed this happens when one goes from ordinary group theory to commutative group theory, and from general field theory to real algebra.

▪ **Page 784 · Predicate logic.** The universality of predicate logic with a single two-argument function follows immediately from the result on page 1156 that it can be used to emulate any two-way multiway system.

▪ **Page 784 · Algebraic axioms.** How universality works with algebraic axioms depends on how those axioms are being used (compare page 1153). What is said in the main text here assumes that they are being used as on page 773—with each variable in effect standing for any object (compare page 1169), and with the axioms being added to predicate logic. The first of these points means that one is concerned with so-called pure group theory—and with finding results valid for all possible groups. The second means that the statements one considers need not just be of the form $\ldots == \ldots$, but can explicitly involve logic; an example is Cayley's theorem

$$a \cdot x == a \cdot y \Rightarrow (x == y \wedge \exists_z a \cdot z == x) \wedge$$
$$a \cdot x == b \cdot x \Rightarrow a == b \wedge (a \cdot b) \cdot x == a \cdot (b \cdot x)$$

With this setup, Alfred Tarski showed in 1946 that any statement in Peano arithmetic can be encoded as a statement in group theory—thus demonstrating that group theory is universal, and that questions about it can be undecidable. This then also immediately follows for semigroup theory and monoid theory. It was shown for ring theory and field

theory by Julia Robinson in 1949. But for commutative group theory it is not the case, as shown by Wanda Szmielew around 1950. And indeed there is a procedure based on quantifier elimination for determining in a finite number of steps whether any statement in commutative group theory can be proved. (Commutative group theory is thus a decidable theory. But as mentioned in the note above, it is not complete—since for example it cannot establish the theorem $a == b$ which states that a group has just one element. It is nevertheless not essentially incomplete—and for example adding the axiom $a == b$ makes it complete.) Real algebra is also not universal (see page 1153), and the same is for example true for finite fields—but not for arbitrary fields.

As discussed on page 1141, word problems for systems such as groups are undecidable. But to set up a word problem in general formally requires going beyond predicate logic, and including axioms from set theory. For a word problem relates not, say, to groups in general, but to a particular group, specified by relations between generators. Within predicate logic one can give the relations as statements, but in effect one cannot specify that no other relations hold. It turns out, however, that undecidability for word problems occurs in essentially the same places as universality for axioms with predicate logic. Thus, for example, the word problem is undecidable for groups and semigroups, but is decidable for commutative groups.

One can also consider using algebraic axioms without predicate logic—as in basic logic or in the operator systems of page 801. And one can now ask whether there is then universality. In the case of semigroup theory there is not. But certainly systems of this type can be universal—since for example they can be set up to emulate any multiway system. And it seems likely that the axioms of ordinary group theory are sufficient to achieve universality.

■ **Page 784 · Set theory.** Any integer n can be encoded as a set using for example `Nest[Union[#, {#}] &, {}, n]`. And from this a statement s in Peano arithmetic (with each variable explicitly quantified) can be translated to a statement in set theory by using

```
Replace[s, {∀_a_ b_ → ∀_a (a ∈ ℕ ⇒ b),
    ∃_a_ b_ → ∃_a (a ∈ ℕ ∧ b)}, {0, ∞}]
```

and then adding the statements below to provide definitions ($\mathbb{N}$ is the set of non-negative integers, $\langle x, y, z\rangle$ is an ordered triple, and $\updownarrow a$ determines whether each triple in a set a is of the form $\langle x, y, f[x, y]\rangle$ specifying a single-valued function).

$a = \mathbb{N} \Leftrightarrow \forall_b ((\emptyset \in b \wedge \forall_c (c \in b \Rightarrow \cup\{c, \{c\}\} \in b)) \Rightarrow a \subseteq b)$
$a = \triangle b \Leftrightarrow a = \cup\{b, \{b\}\}$
$a = \langle b, c, d\rangle \Leftrightarrow a = \{\{\{b, c\}, \{c\}\}, d\}, \{d\}\}$
$\updownarrow a \Leftrightarrow (\forall_b \forall_c \forall_d (\langle b, c, d\rangle \in a \Rightarrow \forall_e (\langle b, c, e\rangle \in a \Rightarrow d = e)) \wedge$ $\forall_b \forall_c ((b \in \mathbb{N} \wedge c \in \mathbb{N}) \Rightarrow \exists_d (d \in \mathbb{N} \wedge \langle b, c, d\rangle \in a)))$
$a = b + c \Leftrightarrow \forall_d ((\updownarrow d \wedge \forall_e \forall_f \forall_g (\langle \triangle e, f, g\rangle \in d \Rightarrow \langle e, f, \triangle g\rangle \in d) \wedge$ $\forall_f \forall_g (\langle \emptyset, f, g\rangle \in d \Rightarrow g = f)) \Rightarrow \langle b, c, a\rangle \in d)$
$a = b \times c \Leftrightarrow \forall_d ((\updownarrow d \wedge \forall_e \forall_f \forall_g (\langle \triangle e, f, g\rangle \in d \Rightarrow \langle e, f, f + g\rangle \in d) \wedge$ $\forall_f \forall_g (\langle \emptyset, f, g\rangle \in d \Rightarrow g = \emptyset)) \Rightarrow \langle b, c, a\rangle \in d)$

This means that set theory can be used to prove any statement that can be proved in Peano arithmetic. But it can also prove other statements—such as Goodstein's result (see note below), and the consistency of arithmetic (see page 1168). An important reason for this is that set theory allows not just ordinary induction over sequences of integers but also transfinite induction over arbitrary ordered sets (see below).

■ **Page 786 · Universal Diophantine equation.** The equation is built up from ones whose solutions are set up to be integers that satisfy particular relations. So for example the equation $a^2 + b^2 == 0$ has solutions that are exactly those integers that satisfy the relation $a == 0 \wedge b == 0$. Similarly, assuming as in the rest of this note that all variables are non-negative, $b == a + c + 1$ has solutions that are exactly those integers that satisfy $a < b$, with c having some allowed value. From various number-theoretical results many relations can readily be encoded as integer equations:

```
(a == 0 ∨ b == 0) ↔ a b == 0

(a == 0 ∧ b == 0) ↔ a + b == 0

a < b ↔ b == a + c + 1

a == Mod[b, c] ↔ (b == a + c d ∧ a < c)

a == Quotient[b, c] ↔ (b == a c + d ∧ d < c)

a == Binomial[b, c] ↔ With[{n = 2^b + 1},
      (n + 1)^b == n^c (a + d n) + e ∧ e < n^c ∧ a < n]

a == b! ↔ a == Quotient[c^b, Binomial[c, b]]

a == GCD[b, c] ↔ (b c > 0 ∧ a d == b ∧ a e == c ∧ a + c f == b g)

a == Floor[b/c] ↔ (a c + d == b ∧ d < c)

PrimeQ[a] ↔ (GCD[(a - 1)!, a] == 1 ∧ a > 1)

a == BitAnd[c, d] ∧ b == BitOr[c, d] ↔
    (σ[c, a] ∧ σ[d, a] ∧ σ[b, c] ∧ σ[b, d] ∧ a + b == c + d) /.
  σ[x_, y_] → Mod[Binomial[x, y], 2] == 1
```

where the last encoding uses the result on page 608. (Note that any variable a can be forced to be non-negative by including an equation $a == w^2 + x^2 + y^2 + z^2$, as on page 910.)

Given an integer a for which `IntegerDigits[a, 2]` gives the cell values for a cellular automaton, a single step of evolution according say to rule 30 is given by

```
BitXor[a, 2 BitOr[a, 2 a]]
```

where (see page 871)

```
BitXor[x, y] == BitOr[x, y] - BitAnd[x, y]
```

and a is assumed to be padded with 0's at each end. The corresponding form for rule 110 is

```
BitXor[BitAnd[a, 2 a, 4 a], BitOr[2 a, 4 a]]
```

The final equation is then obtained from

```
{1 + x4 + x12 == 2^(1+x3) (x1+2 x3), x2 + x13 == 2^x1,
  1 + x5 + x14 == 2^x1, 2^x3 x5 + 2^x1+2 x3 x6 + 2^x1+x3 x15 + x16 == x4,
  1 + x15 + x17 == 2^x3, 1 + x16 + x18 == 2^x3,
  2^1+x3 (1+x1+2 x3) (-1 + x2) - x10 + x11 == 2 x4,
  x7 == BitAnd[x6, 2 x6] ∧ x8 == BitOr[x6, 2 x6],
  x9 == BitAnd[x6, 2 x7] ∧ x19 == BitOr[x6, 2 x7],
  x10 == BitAnd[x9, 2 x8] ∧ x11 == BitOr[x9, 2 x8]}
```

where x_1 through x_4 have the meanings indicated in the main text, and satisfy $x_i \geq 0$. Non-overlapping subsidiary variables are introduced for *BitOr* and *BitAnd*, yielding a total of 79 variables.

Note that it is potentially somewhat easier to construct Diophantine equations to emulate register machines—or arithmetic systems from page 673—than to emulate cellular automata, but exactly the same basic methods can be used.

In the universal equation in the main text variables appear in exponents. One can reduce such an exponential equation to a pure polynomial equation by encoding powers using integer equations. The simplest known way of doing this (see note below) involves a degree 8 equation with 60 variables:

```
a == b^c ↔ α[d, 4 + b e, 1 + z] ∧ α[f, e, 1 + z] ∧
  a == Quotient[d, f] ∧ α[g, 4 + b, 1 + z] ∧ e == 16 g (1 + z)
λ[a_, b_, c_] := Module[{x},
    2 a + x1 == c ∧ (Mod[b - a, c] == 0 ∨ Mod[b + a, c] == 0)]
α[a_, b_, c_] := Module[{x}, x1^2 - b x1 x2 + x2^2 == 1 ∧
  x3^2 - b x3 x4 + x4^2 == 1 ∧ 1 + x4 + x5 == x3 ∧ Mod[x3, x1^2] ==
   0 ∧ 2 x4 + x7 == b x3 ∧ Mod[-b + x8, x7] == 0 ∧
  Mod[-2 + x8, x1] == 0 ∧ x8 - x11 == 3 ∧ x12^2 - x8 x12 x13 +
    x13^2 == 1 ∧ 1 + 2 a + x14 == x1 ∧ λ[a, x12, x7] ∧ λ[c, x12, x1]]
```

(This roughly uses the idea that solutions to Pell equations grow exponentially, so that for example $x^2 == 2y^2 + 1$ has solutions $With[\{u = 3 + 2\sqrt{2}\}, (u^n + u^{-n})/2]$.) From this representation of *Power* the universal equation can be converted to a purely polynomial equation with 2154 variables—which when expanded has 1683150 terms, total degree 16 (average per term 6.8), maximum coefficient 17827424 and *LeafCount* 16540206.

Note that the existence of universal Diophantine equations implies that any problem of mathematics—even, say, the Riemann Hypothesis—can in principle be formulated as a question about the existence of solutions to a Diophantine equation. It also means that given any specific enumeration of polynomials, there must be some universal polynomial u which if fed the enumeration number of a polynomial p, together with an encoding of the values of its variables, will yield the corresponding value of p as a solution to $u == 0$.

▪ **Hilbert's Tenth Problem.** Beginning in antiquity various procedures were developed for solving particular kinds of Diophantine equations (see page 1164). In 1900, as one of his list of 23 important mathematical problems, David Hilbert posed the problem of finding a single finite procedure that could systematically determine whether a solution exists to any specified Diophantine equation. The original proof of Gödel's Theorem from 1931 in effect involves showing that certain logical and other operations can be represented by Diophantine equations—and in the end Gödel's Theorem can be viewed as saying that certain statements about Diophantine equations are unprovable. The notion that there might be universal Diophantine equations for which Hilbert's Tenth Problem would be fundamentally unsolvable emerged in work by Martin Davis in 1953. And by 1961 Davis, Hilary Putnam and Julia Robinson had established that there are exponential Diophantine equations that are universal. Extending this to show that Hilbert's original problem about ordinary polynomial Diophantine equations is unsolvable required proving that exponentiation can be represented by a Diophantine equation, and this was finally done by Yuri Matiyasevich in 1969 (see note above).

By the mid-1970s, Matiyasevich had given a construction for a universal Diophantine equation with 9 variables—though with a degree of about 10^{45}. It had been known since the 1930s that any Diophantine equation can be reduced to one with degree 4—and in 1980 James Jones showed that a universal Diophantine equation with degree 4 could be constructed with 58 variables. In 1979 Matiyasevich also showed that universality could be achieved with an exponential Diophantine equation with many terms, but with only 3 variables. As discussed in the main text I believe that vastly simpler Diophantine equations can also be universal. It is even conceivable that a Diophantine equation with 2 variables could be universal: with one variable essentially being used to represent the program and input, and the other the execution history of the program—with no finite solution existing if the program does not halt.

▪ **Polynomial value sets.** Closely related to issues of solving Diophantine equations is the question of what set of positive values a polynomial can achieve when fed all possible positive integer values for its variables. A polynomial with a single variable must always yield either be a finite set, or a simple polynomial progression of values. But already the sequence of values for $x^2 y - x y^3$ or even $x(y^2 + 1)$ seem quite complicated. And for example from the fact that $x^2 == y^2 + (x y \pm 1)$ has solutions *Fibonacci[n]* it follows that

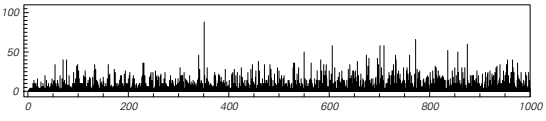
the positive values of $(2-(x^2-y^2-xy)^2)x$ are just *Fibonacci[n]* (achieved when *{x, y}* is *Fibonacci[{n, n-1}]*). This is the simplest polynomial giving *Fibonacci[n]*, and there are for example no polynomials with 2 variables, up to 4 terms, total degree less than 4, and integer coefficients between -2 and +2, that give any of 2^n, 3^n or *Prime[n]*. Nevertheless, from the representation for *PrimeQ* in the note above it has been shown that the positive values of a particular polynomial with 26 variables, 891 terms and total degree 97 are exactly the primes. (Polynomials with 42 variables and degree 5, and 10 variables and degree 10^{45}, are also known to work, while it is known that one with 2 variables cannot.) And in general the existence of a universal Diophantine equation implies that any set obtained by any finite computation must correspond to the positive values of some polynomial. The analog of doing a long computation to find a result is having to go to large values of variables to find a positive polynomial value. Note that one can imagine, say, emulating the evolution of a cellular automaton by having the t^{th} positive value of a polynomial represent the t^{th} step of evolution. That universality can be achieved just in the positive values of a polynomial is already remarkable. But I suspect that in the end it will take only a surprisingly simple polynomial, perhaps with just three variables and fairly low degree.

(See also page 1165.)

▪ **Statements in Peano arithmetic.** Examples include:

- $\sqrt{2}$ is irrational:
  $\neg\, \exists_a (\exists_b (b \neq 0 \wedge a \times a == (\Delta\Delta 0) \times (b \times b)))$
- There are infinitely many primes of the form $n^2 + 1$:
  $\neg\, \exists_n (\forall_c (\exists_a (\exists_b (n + c) \times (n + c) + \Delta 0 == (\Delta\Delta a) \times (\Delta\Delta b))))$
- Every even number (greater than 2) is the sum of two primes (Goldbach's Conjecture; see page 135):
  $\forall_a (\exists_b (\exists_c ((\Delta\Delta 0) \times (\Delta\Delta a) == b + c \wedge \forall_d (\forall_e (\forall_f ((f == (\Delta\Delta d) \times (\Delta\Delta e) \vee f == \Delta 0) \Rightarrow (f \neq b \wedge f \neq c)))))))$

The last two statements have never been proved true or false, and remain unsolved problems of number theory. The picture shows spacings between n for which $n^2 + 1$ is prime.



▪ **Transfinite numbers.** For most mathematical purposes it is quite adequate just to have a single notion of infinity, usually denoted ∞. But as Georg Cantor began to emphasize in the 1870s, it is possible to distinguish different levels of infinity. Most of the details of this have not been widely used in typical mathematics, but they can be helpful in studying foundational issues. Cantor's theory of ordinal numbers is based on the idea that every integer must have a successor. The next integer after all of the ordinary ones—the first infinite integer—is given the name ω. In Cantor's theory $\omega + 1$ is still larger (though $1 + \omega$ is not), as are $2\,\omega$, ω^2 and ω^ω. Any arithmetic expression involving ω specifies an ordinal number—and can be thought of as corresponding to a set containing all integers up to that number. The ordinary axioms of arithmetic do not apply, but there are still fairly straightforward rules for manipulating such expressions. In general there are many different expressions that correspond to a given number, though there is always a unique Cantor normal form—essentially a finite sequence of digits giving coefficients of descending powers of ω. However, not all infinite integers can be represented in this way. The first one that cannot is ϵ_0, given by the limit $\omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$, or effectively *Nest[ω^# &, ω, ω]*. ϵ_0 is the smallest solution to $\omega^\epsilon == \epsilon$. Subsequent solutions ($\epsilon_1$, ..., ϵ_ω, ..., ϵ_{ϵ_0}, ...) define larger ordinals, and one can go on until one reaches the limit $\epsilon_{\epsilon_{\epsilon_{\cdot^{\cdot^{\cdot}}}}}$, which is the first solution to $\epsilon_\alpha == \alpha$. Giving this ordinal a name, one can then go on again, until eventually one reaches another limit. And it turns out that in general one in effect has to introduce an infinite sequence of names in order to be able to specify all transfinite integers. (Naming a single largest or "absolutely infinite" integer is never consistent, since one can always then talk about its successor.) As Cantor noted, however, even this only allows one to reach the lowest class of transfinite numbers—in effect those corresponding to sets whose size corresponds to the cardinal number $\aleph_0$. Yet as discussed on page 1127, one can also consider larger cardinal numbers, such as $\aleph_1$, considered in connection with the number of real numbers, and so on. And at least for a while the ordinary axioms of set theory can be used to study the sets that arise.

▪ **Growth rates.** One can characterize most functions by their ultimate rates of growth. In basic mathematics these might be n, $2\,n$, $3\,n$, ... or n^2, n^3, ..., or 2^n, 3^n, ..., or 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... To go further one begins by defining an analog to the Ackermann function of page 906:

f[1][n_] = 2 n; f[s_][n_] := Nest[f[s - 1], 1, n]

f[2][n] is then 2^n, *f[3]* is iterated power, and so on. Given this one can now form the "diagonal" function

f[ω][n_] := f[n][n]

and this has a higher growth rate than any of the *f[s][n]* with finite *s*. This higher growth rate is indicated by the transfinite index ω. And in direct analogy to the transfinite numbers

discussed above one can then in principle form a hierarchy of functions using operations like

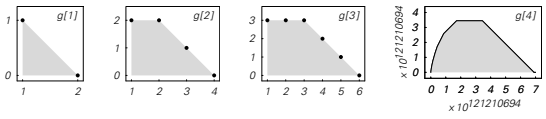
```
f[ω + s][n_] := Nest[f[ω + s - 1], 1, n]
```

together with diagonalization at limit ordinals. In practice, however, it gets more and more difficult to determine that the functions defined in this way actually in a sense halt and yield definite values—and indeed for $f[\epsilon_0]$ this can no longer be proved using the ordinary axioms of arithmetic (see below). Yet it is still possible to define functions with even more rapid rates of growth. An example is the so-called busy beaver function (see page 1144) that gives the maximum number of steps that it takes for any Turing machine of size n to halt when started from a blank tape. In general this function must grow faster than any computable function, and is not itself computable.

▪ **Page 787 · Unprovable statements.** After the appearance of Gödel's Theorem a variety of statements more or less directly related to provability were shown to be unprovable in Peano arithmetic and certain other axiom systems. Starting in the 1960s the so-called method of forcing allowed certain kinds of statements in strong axiom systems—like the Continuum Hypothesis in set theory (see page 1155)—to be shown to be unprovable. Then in 1977 Jeffrey Paris and Leo Harrington showed that a variant of Ramsey's Theorem (see page 1068)—a statement that is much more directly mathematical—is also unprovable in Peano arithmetic. The approach they used was in essence based on thinking about growth rates—and since the 1970s almost all new examples of unprovability have been based on similar ideas. Probably the simplest is a statement shown to be unprovable in Peano arithmetic by Laurence Kirby and Jeff Paris in 1982: that certain sequences $g[n]$ defined by Reuben Goodstein in 1944 are of limited length for all n, where

```
g[n_] := Map[First, NestWhileList[
    {f[#] - 1, Last[#] + 1} &, {n, 3}, First[#] > 0 &]]
f[{0, _}] = 0; f[{n_, k_}] := Apply[Plus, MapIndexed[#1
    k^f[{#2[[1]] - 1, k}] &, Reverse[IntegerDigits[n, k - 1]]]]
```

As in the pictures below, $g[1]$ is $\{1, 0\}$, $g[2]$ is $\{2, 2, 1, 0\}$ and $g[3]$ is $\{3, 3, 3, 2, 1, 0\}$. $g[4]$ increases quadratically for a long time, with only element $3 \times 2^{402653211} - 2$ finally being 0. And the point is that in a sense $Length[g[n]]$ grows too quickly for its finiteness to be provable in general in Peano arithmetic.



The argument for this as usually presented involves rather technical results from several fields. But the basic idea is roughly just to set up a correspondence between elements of $g[n]$ and possible proofs in Peano arithmetic—then to use the fact that if one knew that $g[n]$ always terminated this would establish the validity of all these proofs, which would in turn prove the consistency of arithmetic—a result which is known to be unprovable from within arithmetic.

Every possible proof in Peano arithmetic can in principle be encoded as an ordinary integer. But in the late 1930s Gerhard Gentzen showed that if proofs are instead encoded as ordinal numbers (see note above) then any proof can validly be reduced to a preceding one just by operations in logic. To cover all possible proofs, however, requires going up to the ordinal ϵ_0. And from the unprovability of consistency one can conclude that this must be impossible using the ordinary operation of induction in Peano arithmetic. (Set theory, however, allows transfinite induction—essentially induction on arbitrary sets—letting one reach such ordinals and thus prove the consistency of arithmetic.) In constructing $g[n]$ the integer n is in effect treated like an ordinal number in Cantor normal form, and a sequence of numbers that should precede it are found. That this sequence terminates for all n is then provable in set theory, but not Peano arithmetic—and in effect $Length[g[n]]$ must grow like $f[\epsilon_0][n]$.)

In general one can imagine characterizing the power of any axiom system by giving a transfinite number κ which specifies the first function $f[\kappa]$ (see note above) whose termination cannot be proved in that axiom system (or similarly how rapidly the first example of y must grow with x to prevent $\exists_y p[x, y]$ from being provable). But while it is known that in Peano arithmetic $\kappa = \epsilon_0$, quite how to describe the value of κ for, say, set theory remains unknown. And in general I suspect that there are a vast number of functions with simple definitions whose termination cannot be proved not just because they grow too quickly but instead for the more fundamental reason that their behavior is in a sense too complicated.

Whenever a general statement about a system like a Turing machine or a cellular automaton is undecidable, at least some instances of that statement encoded in an axiom system must be unprovable. But normally these tend to be complicated and not at all typical of what arise in ordinary mathematics. (See page 1167.)

▪ **Encodings of arithmetic.** Statements in arithmetic are normally written in terms of $+$, $\times$ and Δ (and logical operations). But it turns out also to be possible to encode such statements in terms of other basic operations. This was for example done by Julia Robinson in 1949 with Δ (or $a + 1$) and $Mod[a, b] == 0$. And in the 1990s Ivan Korec and others

showed that it could be done just with *Mod[Binomial[a + b, a], k]* with $k = 6$ or any product of primes—and that it could not be done with k a prime or prime power. These operations can be thought of as finding elements in nested Pascal's triangle patterns produced by k-color additive cellular automata. Korec showed that finding elements in the nested pattern produced by the $k = 3$ cellular automaton with rule *{{1, 1, 3}, {2, 2, 1}, {3, 3, 2}}[[#1, #2]] &* (compare page 886) was also enough.

■ **Page 788 · Infinity.** See page 1162.

■ **Page 789 · Diophantine equations.** If variables appear only linearly, then it is possible to use *ExtendedGCD* (see page 944) to find all solutions to any system of Diophantine equations—or to show that none exist. Particularly from the work of Carl Friedrich Gauss around 1800 there emerged a procedure to find solutions to any quadratic Diophantine equation in two variables—in effect by reduction to the Pell equation $x^2 == a y^2 + 1$ (see page 944), and then computing *ContinuedFraction[$\sqrt{a}$]*. The minimal solutions can be large; the largest ones for successive coefficient sizes are given below. (With size s coefficients it is for example known that the solutions must always be less than $(14 s)^{5 s}$.).

1	$1 + x + x^2 + y - x y == 0$	{x == 2, y == 7}
2	$1 + 2 x + 2 x^2 + 2 y + x y - 2 y^2 == 0$	{x == 687, y == 881}
3	$2 + 2 x + 3 x^2 + 3 y + x y - y^2 == 0$	{x == 545759, y == 1256763}
4	$-4 - x + 4 x^2 - y - 3 x y - 4 y^2 == 0$	{x == 251996202018, y == 174633485974}

There is a fairly complete theory of homogeneous quadratic Diophantine equations with three variables, and on the basis of results from the early and mid-1900s a finite procedure should in principle be able to handle quadratic Diophantine equations with any number of variables. (The same is not true of simultaneous quadratic Diophantine equations, and indeed with a vector x of just a few variables, a system $m . x^2 == a$ of such equations could quite possibly show undecidability.)

Ever since antiquity there have been an increasing number of scattered results about Diophantine equations involving higher powers. In 1909 Axel Thue showed that any equation of the form *p[x, y] == a*, where *p[x, y]* is a homogeneous irreducible polynomial of degree at least 3 (such as $x^3 + x y^2 + y^3$) can have only a finite number of integer solutions. (He did this by formally factoring *p[x, y]* into terms $x - \alpha_i y$, then looking at rational approximations to the algebraic numbers α_i.) In 1966 Alan Baker then proved an explicit upper bound on such solutions, thereby establishing that in principle they can be found by a finite search procedure. (The proof is based on having bounds for how close to zero *Sum[α_i Log[α_j], i, j]* can be for independent algebraic numbers α_k.) His bound was roughly *Exp[(c s)$^{10^5}$]*—but later work in essence reduced this, and by the 1990s practical algorithms were being developed. (Even with a bound of 10^{100}, rational approximations to real number results can quickly give the candidates that need to be tested.)

Starting in the late 1800s and continuing ever since a series of progressively more sophisticated geometric and algebraic views of Diophantine equations have developed. These have led for example to the 1993 proof of Fermat's Last Theorem and to the 1983 Faltings theorem (Mordell conjecture) that the topology of the algebraic surface formed by allowing variables to take on complex values determines whether a Diophantine equation has only a finite number of rational solutions—and shows for example that this is the case for any equation of the form $x^n == a y^n + 1$ with $n > 3$. Extensive work has been done since the early 1900s on so-called elliptic curve equations such as $x^2 == a y^3 + b$ whose corresponding algebraic surface has a single hole (genus 1). (A crucial feature is that given any two rational solutions to such equations, a third can always be found by a simple geometrical construction.) By the 1990s explicit algorithms for such equations were being developed—with bounds on solutions being found by Baker's method (see above). In the late 1990s similar methods were applied to superelliptic (e.g. $x^n == p[y]$) and hyperelliptic (e.g. $x^2 == p[y]$) equations involving higher powers, and it now at least definitely seems possible to handle any two-variable cubic Diophantine equation with a finite procedure. Knowing whether Baker's method can be made to work for any particular class of equations involves, however, seeing whether certain rather elaborate algebraic constructions can be done—and this may perhaps in general be undecidable. Most likely there are already equations of degree 4 where Baker's method cannot be used—perhaps ones like $x^3 == y^4 + x y + a$. But in recent years there have begun to be results by other methods about two-variable Diophantine equations, giving, for example, general upper bounds on the number of possible solutions. And although this has now led to the assumption that all two-variable Diophantine equations will eventually be resolved, based on the results of this book I would not be surprised if in fact undecidability and universality appeared in such equations—even perhaps at degree 4 with fairly small coefficients.

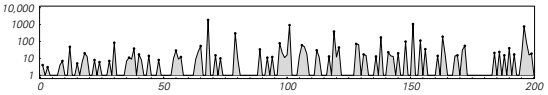
The vast majority of work on Diophantine equations has been for the case of two variables (or three for some homogeneous equations). No clear analog of Baker's method is known beyond two variables, and my suspicion is that with three variables undecidability and universality may already be present even in cubic equations.

As mentioned in the main text, proving that even simple specific Diophantine equations have no solutions can be very difficult. Obvious methods involve for example showing that no solutions exist for real variables, or for variables reduced modulo some n. (For quadratic equations Hasse's Principle implies that if no solutions exist for any n then there are no solutions for ordinary integers—but a cubic like $3x^3 + 4y^3 + 5z^3 == 0$ is a counterexample.) If one can find a bound on solutions—say by Baker's method—then one can also try to show that no values below this bound are actually solutions. Over the history of number theory the sophistication of equations for which proofs of no solutions can be given has gradually increased—though even now it is state of the art to show say that $x == y == 1$ is the only solution to $x^2 == 3y^4 - 2$.

Just as for all sorts of other systems with complex behavior, some idea of overall properties of Diophantine equations can be found on the basis of an approximation of perfect randomness. Writing equations in the form $p[x_1, x_2, \ldots, x_n] == 0$ the distribution of values of p will in general be complicated (see page 1161), but as a first approximation one can try taking it to be purely random. (Versions of this for large numbers of variables are validated by the so-called circle method from the early 1900s.) If p has total degree d then with $x_i < x$ the values of *Abs[p]* will range up to about x^d. But with n variables the number of different cases sampled for $x_i < x$ will be x^n. The assumption of perfect randomness then suggests that for $d < n$, more and more cases with $p == 0$ will be seen as x increases, so that the equation will have an infinite number of solutions. For $d > n$, on the other hand, it suggests that there will often be no solutions, and that any solutions that exist will usually be small. In the boundary case $d == n$ it suggests that even for arbitrarily large x an average of about one solution should exist—suggesting that the smallest solution may be very large, and presumably explaining the presence of so many large solutions in the $n = d = 2$ and $n = d = 3$ examples in the main text. Note that even though large solutions may be rare when $d > n$ they must always exist in at least some cases whenever there is undecidability and universality in a class of equations. (See also page 1161.)

If one wants to enumerate all possible Diophantine equations there are many ways to do this, assigning different weights to numbers of variables, and sizes of coefficients and of exponents. But with several ways I have tried, it seems that of the first few million equations, the vast majority have no solutions—and this can in most cases be established by fairly elementary methods that are presumably within Peano arithmetic. When solutions do exist, most are fairly small. But as one continues the enumeration there are increasingly a few equations that seem more and more difficult to handle.

- **Page 790 · Properties.** (All variables are assumed positive.)
  - $2x + 3y == a$. There are *Ceiling[a/2] + Ceiling[2 a/3] - (a + 1)* solutions, the one with smallest x being *{Mod[2 a + 2, 3] + 1, 2 Floor[(2 a + 2)/3] - (a + 2)}*. Linear equations like this were already studied in antiquity. (Compare page 915.)
  - $x^2 == y^2 + a$. Writing a in terms of distinct factors as $r\,s$, *{r + s, r - s}/2* gives a solution if it yields integers—which happens when *Abs[a] > 4* and *Mod[a, 4] ≠ 2*.
  - $x^2 == a\,y^2 + 1$ (Pell equation). As discussed on page 944, whenever a is not a perfect square, there are always an infinite number of solutions given in terms of *ContinuedFraction[$\sqrt{a}$]*. Note that even when the smallest solution is not very large, subsequent solutions can rapidly get large. Thus for example when $a = 13$, the second solution is already *{842401, 233640}*.
  - $x^2 == y^3 + a$ (Mordell equation). First studied in the 1600s, a complete theory of this so-called elliptic curve equation was only developed in the late 1900s—using fairly sophisticated algebraic number theory. The picture below shows as a function of a the minimum x that solves the equation. For $a = 68$, the only solution is $x = 1874$; for $a = 1090$, it is $x = 149651610621$. The density of cases with solutions gradually thins out as a increases (for $0 < a \le 10000$ there are 2468 such cases). There are always only a finite number of solutions (for $0 < a \le 10000$ the maximum is 12, achieved for $a = 8900$).



  - $x^2 == a\,y^3 + 1$. Also an elliptic curve equation.
  - $x^3 == y^4 + x\,y + a$. For most values of a (including specifically $a = 1$) the continuous version of this equation defines a surface of genus 3, so there are at most a finite number of integer solutions. (An equation of degree d generically defines a surface of genus $1/2\,(d-1)(d-2)$.) Note that $x^3 == y^4 + a$ is equivalent to $x^3 == z^2 + a$ by a simple substitution.
  - $x^2 == y^5 + a\,y + 3$. The second smallest solution to $x^2 == y^5 + 5y + 3$ is *{45531, 73}*. As for the equations above, there are always at most a finite number of integer solutions.
  - $x^3 + y^3 == z^2 + a$. For the homogenous case $a = 0$ the complete solution was found by Leonhard Euler in 1756.

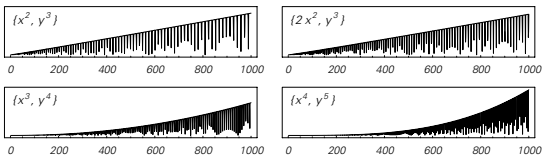
- $x^3 + y^3 == z^3 + a$. No solutions exist when $a = 9n \pm 4$; for $a = n^3$ or $2n^3$ infinite families of solutions are known. Particularly in its less strict form $x^3 + y^3 + z^3 == a$ with x, y, z positive or negative the equation was mentioned in the 1800s and again in the mid-1900s; computer searches for solutions were begun in the 1960s, and by the mid-1990s solutions such as $\{283059965, 2218888517, 2220422932\}$ for the case $a = -30$ had been found. Any solution to the difficult case $x^3 + y^3 == z^3 - 3$ must have $Mod[x, 9] == Mod[y, 9] == Mod[z, 9]$. (Note that $x^2 + y^2 + z^2 == a$ always has solutions except when $a = 4^s(8n + 7)$, as mentioned on page 135.)

■ **Large solutions.** A few other 2-variable equations with fairly large smallest solutions are:

- $x^3 == 3y^3 - xy + 63$: $\{7149, 4957\}$
- $x^4 == y^3 + 2xy - 2y + 81$: $\{19674, 531117\}$
- $x^4 == 5y^3 + xy + y - 8x$: $\{69126, 1659072\}$

The equation $x^x y^y == z^z$ is known to have smallest non-trivial solution $\{2985984, 1679616, 4478976\}$.

■ **Nearby powers.** One can potentially find integer equations with large solutions but small coefficients by looking say for pairs of integer powers close in value. The pictures below show what happens if one computes x^m and y^n for many x and y, sorts these values, then plots successive differences. The differences are trivially zero when $x = s^n$, $y = s^m$. Often they are large, but surprisingly small ones can sometimes occur (despite various suggestions from the so-called ABC conjecture). Thus, for example, $5853886516781223^3 - 1641843$ is a perfect square, as found by Noam Elkies in 1998. (Another example is $55^5 - 22434^2 == 19$.)



■ **Page 791 · Unsolved problems.** Problems in number theory that are simple to state (say in the notation of Peano arithmetic) but that so far remain unsolved include:

- Is there any odd number equal to the sum of its divisors? (Odd perfect number; 4th century BC) (See page 911.)
- Are there infinitely many primes that differ by 2? (Twin Prime Conjecture; 1700s?) (See page 909.)
- Is there a cuboid in which all edges and all diagonals are of integer length? (Perfect cuboid; 1719)
- Is there any even number which is not the sum of two primes? (Goldbach's Conjecture; 1742) (See page 135.)
- Are there infinitely many primes of the form $n^2 + 1$? (Quadratic primes; 1840s?) (See page 1162.)
- Are there infinitely many primes of the form $2^{2^n} + 1$? (Fermat primes; 1844)
- Are there no solutions to $x^m - y^n == 1$ other than $3^2 - 2^3 == 1$? (Catalan's Conjecture; 1844)
- Can every integer not of the form $9n \pm 4$ be written as $a^3 \pm b^3 \pm c^3$? (See note above.)
- How few n^{th} powers need be added to get any given integer? (Waring's Problem; 1770)

(See also Riemann Hypothesis on page 918.)

■ **Page 791 · Fermat's Last Theorem.** That $x^n + y^n == z^n$ has no integer solutions for $n > 2$ was suggested by Pierre Fermat around 1665. Fermat proved this for $n = 4$ around 1660; Leonhard Euler for $n = 3$ around 1750. It was proved for $n = 5$ and $n = 7$ in the early 1800s. Then in 1847 Ernst Kummer used ideas of factoring with algebraic integers to prove it for all $n < 37$. Extensions of this method gradually allowed more cases to be covered, and by the 1990s computers had effectively given proofs for all n up to several million. Meanwhile, many connections had been found between the general case and other areas of mathematics—notably the theory of elliptic curves. And finally around 1995, building on extensive work in number theory, Andrew Wiles managed to give a complete proof of the result. His proof is long and complicated, and relies on sophisticated ideas from many areas of mathematics. But while the statement of the proof makes extensive use of concepts from areas like set theory, it seems quite likely that in the end a version of it could be given purely in terms of Peano arithmetic. (By the 1970s it had for example been shown that many classic proofs with a similar character in analytic number theory could at least in principle be carried out purely in Peano arithmetic.)

■ **Page 791 · More powerful axioms.** If one looks for example at progressively more complicated Diophantine equations then one can expect that one will find examples where more and more powerful axiom systems are needed to prove statements about them. But my guess is that almost as soon as one reaches cases that cannot be handled by Peano arithmetic one will also reach cases that cannot be handled by set theory or even by still more powerful axiom systems.

Any statement that one can show is independent of the Peano axioms and at least not inconsistent with them one can potentially consider adding as a new axiom. Presumably it is best to add axioms that allow the widest range of new

statements to be proved. But I strongly suspect that the set of statements that cannot be proved is somehow sufficiently fragmented that adding a few new axioms will actually make very little difference.

In set theory (see page 1155) a whole sequence of new axioms have historically been added to allow particular kinds of statements to be proved. And for several decades additional so-called large cardinal axioms have been discussed, that in effect state that sets exist larger than any that can be reached with the current axioms of set theory. (As discussed on page 816 any axiom system that is universal must in principle be able to prove any statement that can be proved in any axiom system—but not with the kinds of encodings normally considered in mathematical logic.)

It is notable, however, that if one looks at classic theorems in mathematics many can actually be derived from remarkably weak axioms. And indeed the minimal axioms needed to obtain most of mathematics as it is now practiced are probably much weaker than those on pages 773 and 774.

(If one considers for example theorems about computational issues such as whether Turing machines halt, then it becomes inevitable that to cover more Turing machines one needs more axioms—and to cover all possible machines one needs an infinite set of axioms, that cannot even be generated by any finite set of rules.)

▪ **Higher-order logics.** In ordinary predicate—or so-called first-order—logic the objects x that $\forall_x$ and $\exists_x$ range over are variables of the kind used as arguments to functions (or predicates) such as $f[x]$. To set up second-order logic, however, one imagines also being able to use $\forall_f$ and $\exists_f$ where f is a function (say the head of $f[x]$). And then in third-order logic one imagines using $\forall_g$ and $\exists_g$ where g appears in $g[f][x]$.

Early formulations of axiom systems for mathematics made little distinction between first- and second-order logic. The theory of types used in *Principia Mathematica* introduced some distinction, and following the proof of Gödel's Completeness Theorem for first-order logic in 1930 (see page 1152) standard axiom systems for mathematics (as given on pages 773 and 774) began to be reformulated in first-order form, with set theory taking over many of the roles of second-order logic.

In current mathematics, second-order logic is sometimes used at the level of notation, but almost never in its full form beyond. And in fact with any standard computational system it can never be implemented in any explicit way. For even to enumerate theorems in second-order logic is in general impossible for a system like a Turing machine unless one assumes that an oracle can be added. (Note however that this is possible in Henkin versions of higher-order logic that allow only limited function domains.)

▪ **Truth and incompleteness.** In discussions of the foundations of mathematics in the early 1900s it was normally assumed that truth and provability were in a sense equivalent—so that all true statements could in principle be reached by formal processes of proof from fixed axioms (see page 782). Gödel's Theorem showed that there are statements that can never be proved from given axioms. Yet often it seemed inevitable just from the syntactic structure of statements (say as well-formed formulas) that each of them must at some level be either true or false. And this led to the widespread claim that Gödel's Theorem implies the existence of mathematical statements that are true but unprovable—with their negations being false but unprovable. Over the years this often came to be assigned a kind of mystical significance, mainly because it was implicitly assumed that somehow it must still ultimately be possible to know whether any given statement is true or false. But the Principle of Computational Equivalence implies that in fact there are all sorts of statements that simply cannot be decided by any computational process in our universe. So for example, it must in some sense be either true or false that a given Turing machine halts with given input—but according to the Principle of Computational Equivalence there is no finite procedure in our universe through which we can guarantee to know which of these alternatives is correct.

In some cases statements can in effect have default truth values—so that showing that they are unprovable immediately implies, say, that they must be true. An example in arithmetic is whether some integer equation has no solution. For if there were a solution, then given the solution it would be straightforward to give a proof that it is correct. So if it is unprovable that there is no solution, then it follows that there must in fact be no solution. And similarly, if it could be shown for example that Goldbach's Conjecture is unprovable then it would follow that it must be true, for if it were false then there would have to be a specific number which violates it, and this could be proved. Not all statements in mathematics have this kind of default truth value. And thus for example the Continuum Hypothesis in set theory is unprovable but could be either of true or false: it is just independent of the axioms of set theory. In computational systems, showing that it is unprovable that a given Turing machine halts with given input immediately implies that in fact it must not halt. But showing that it is unprovable whether a Turing machine halts with every input (a Π_2 statement in the notation of page 1139) does not immediately imply anything about whether this is in fact true or false.

■ **Page 793 · Generalization in mathematics.** Systems that have evolved from the basic notion of numbers provide a characteristic example of the process of progressive generalization in mathematics. The main such systems and their dates of earliest known reasonably formalized use have been (see also page 901): positive integers (before 10,000 BC), rationals (3000 BC), square roots (2000 BC), other roots (1800 BC), all integers (600 AD, 1600s), decimals (950 AD), complex numbers (1500s, 1800s), polynomials (1591), infinitesimals (1635), algebraic numbers (1744), quaternions (1843), Grassmann algebra (1844), ideals (1844, 1871), octonions (~1845), Boolean algebra (1847), fields (1850s, 1871), matrices (1858), associative algebras (1870), axiomatic real numbers (1872), vectors (1881), transfinite ordinals (1883), transfinite cardinals (1883), operator calculus (1880s), Boolean algebras (1890), algebraic number fields (1893), rings (1897), p-adic numbers (1897), non-Archimedean fields (1899), q-numbers (1926), non-standard integers (1930s), non-standard reals (hyperreals) (1960), interval arithmetic (1968), fuzzy arithmetic (1970s), surreal numbers (1970s). New systems have usually been introduced in connection with extending the domains of particular existing operations. But in almost all cases the systems are set up so as to preserve as many theorems as possible—a notion that was for example made explicit in the Principle of Permanence discussed by George Peacock in 1830 and extended by Hermann Hankel in 1869.

■ **Page 794 · Cellular automaton axioms.** The first 4 axioms are general to one-dimensional cellular automata. The next 8 are specific to rule 110. The final 3 work whenever patterns are embedded in a background of white cells. The universality of rule 110 presumably implies that the axiom system given is universal. (A complete proof would require handling various issues about boundary conditions.)

If the last 2 axioms are dropped any statement can readily be proved true or false essentially just by running rule 110 for a finite number of steps equal to the number of nested ↓ plus ⟨…⟩ in the statement. In practice, a large number of steps can however be required. As an example the statement

O■ ⇒ (∃$_a$ (∃$_b$ *O a* ◇ (■ ◇ (■ ◇ (■ ◇ (□ ◇
(■ ◇ (■ ◇ (■ ◇ (□ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (□ ◇
(□ ◇ (□ ◇ (■ ◇ (□ ◇ (□ ◇ (■ ◇ (■ ◇ (□ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇ (■ ◇
(□ ◇ (□ ◇ (□ ◇ (■ ◇ (□ ◇ *b*))))))))))))))))))))))))))))))))))

asserts that a particular localized structure occurs in the evolution of rule 110 from a single black cell. But page 38 shows that this happens for the first time after 2867 steps. (A proof of this without lemmas would probably have to be of length at least 32,910,300.)

The axioms as they are stated apply to any rule 110 evolution, regardless of initial conditions. One can establish that the statement at the bottom on the right cannot be proved either true or false from the axioms by showing that it is true for some initial conditions and false for others. Note from page 279 that the sequence □■□■□ cannot occur in rule 110 evolution except as an initial condition. So this means that the statement is false if the initial condition is ■ and true if the initial condition is □■□■□.

■ **Practical programs.** Any equivalence between programs in a programming language can be thought of as a theorem. Simple examples in *Mathematica* include:

First[Prepend[p, q]] === q

Join[Join[p, q], r] === Join[p, Join[q, r]]

Partition[Union[p], 1] === Split[Union[p]]

One can set up axiom systems say by combining definitions of programming languages with predicate logic (as done by John McCarthy for LISP in 1963). And for programs whose structure is simple enough it has sometimes been possible to prove theorems useful for optimization or verification. But in the vast majority of cases this has been essentially impossible in practice. And I suspect that this is a reflection of widespread fundamental unprovability. In setting up programs with specific purposes there is inevitably some computational reducibility (see page 828). But I suspect that enough computational irreducibility usually remains to make unprovability common when one asks about all possible forms of behavior of the program.

■ **Page 796 · Rules.** The examples shown here (roughly in order of increasing complexity) correspond respectively to cases (a), (k), (b), (q), (p), (r), (o), (d) on page 798.

■ **Page 797 · Consistency.** Any axiom system that is universal can represent the statement that the system is consistent. But normally such a statement cannot be proved true or false within the system itself. And thus for example Kurt Gödel showed this in 1931 for Peano arithmetic (in his so-called second incompleteness theorem). In 1936, however, Gerhard Gentzen showed that the axioms of set theory imply the consistency of Peano arithmetic (see page 1160). In practical mathematics set theory is always taken to be consistent, but to set up a proof of this would require axioms beyond set theory.

■ **Page 798 · Properties.** For most of the rules shown, there ultimately turn out to be quite easy characterizations of what strings can be produced.

- (a) At step t, the only new string produced is the one containing t black elements.

- (b) All strings of length n containing exactly one black cell are produced—after at most $2n-1$ steps.
- (c) All strings containing even-length runs of white cells are produced.
- (d) The set of strings produced is complicated. The last length 4 string produced is □□□■, after 16 steps; the last length 6 one is □□□□□■, after 26 steps.
- (e) All strings that begin with a black element are produced.
- (f) All strings that end with a white element but contain at least one black element, or consist of all white elements ending with black, are produced. Strings of length n take n steps to produce.
- (g) The same strings as in (f) are produced, but now a string of length n with m black elements takes $n+m-1$ steps.
- (h) All strings appear in which the first run of black elements is of length 1; a string of length n with m black elements appears after $n+m-1$ steps.
- (i) All strings containing an odd number of black elements are produced; a string of length n with m black cells occurs at step $n+m-1$.
- (j) All strings that end with a black element are produced.
- (k) Above length 1, the strings produced are exactly those starting with a white element. Those of length n appear after at most $3n-3$ steps.
- (l) The same strings as in (k) are produced, taking now at most $2n+1$ steps.
- (m) All strings beginning with a black element are produced, after at most $3n+1$ steps.
- (n) The set of strings produced is complicated, and seems to include many but not all that do not end with ■□.
- (o) All strings that do not end in ■□ are produced.
- (p) All strings are produced, except ones in which every element after the first is white. ■□□■ takes 14 steps.
- (q) All strings are produced, with a string of length n with m white elements taking $n+2m$ steps.
- (r) All strings are ultimately produced—which is inevitable after the lemmas ■ → ■■ and ■ → □ appear at steps 12 and 13. (See the first rule on page 778.)

■ **Page 800 · Non-standard arithmetic.** Goodstein's result from page 1163 is true for all ordinary integers. But since it is independent of the axioms of arithmetic there must be objects that still satisfy the axioms but for which it is false. It turns out however that any such objects must in effect be infinite. For any set of objects that satisfy the axioms of arithmetic must include all finite ordinary integers, since each of these can be reached just by using $\triangle$ repeatedly. And the axioms then turn out to imply that any additional objects must be larger than all these integers—and must therefore be infinite. But for any such truly infinite objects operations like $+$ and $\times$ cannot be computed by finite procedures, making it difficult to describe such objects in an explicit way. Ever since the work of Thoralf Skolem in 1933 non-standard models of arithmetic have been discussed, particularly in the context of ultrafilters and constructs like infinite trees. (See also page 1172.)

■ **Page 800 · Reduced arithmetic.** (See page 1152.) Statements that can be proved with induction but are not provable only with Robinson's axioms are: $x \neq \triangle x$; $x+y == y+x$; $x+(y+z) == (x+y)+z$; $0+x == x$; $\exists_x (\triangle x + y == z \Rightarrow y \neq z)$; $x \times y == y \times x$; $x \times (y \times z) == (x \times y) \times z$; $x \times (y+z) == x \times y + x \times z$.

■ **Page 800 · Generators and relations.** In the axiom systems of page 773, a single variable can stand for any element—much like a *Mathematica* pattern object such as `x_`. In studying specific instances of objects like groups one often represents elements as products of constants or generators, and then for example specifies the group by giving relations between these products. In traditional mathematical notation such relations normally look just like ordinary axioms, but in fact the variables that appear in them are now assumed to be literal objects—like x in *Mathematica*—that are generically taken to be unequal. (Compare page 1159.)

■ **Page 801 · Comparison to multiway systems.** Operator systems are normally based on equations, while multiway systems are based on one-way transformations. But for multiway systems where each rule $p \to q$ is accompanied by its reverse $q \to p$, and such pairs are represented say by `"AAB" ↔ "BBAA"`, an equivalent operator system can immediately be obtained either from

```
Apply[Equal,
  Map[Fold[#2[#1] &, x, Characters[#]] &, rules, {2}], {1}]
```

or from (compare page 1172)

```
Append[Apply[Equal,
    Map[(Fold[f, First[#], Rest[#]] &)[Characters[#]] &,
      rules, {2}], {1}], f[f[a, b], c] == f[a, f[b, c]]]
```

where now objects like `"A"` and `"B"` are treated as constants—essentially functions with zero arguments. With slightly more effort multiway systems with ordinary one-way rules can also be converted to operator systems. Converting from operator systems to multiway systems is more difficult, though ultimately always possible (see page 1156).

As discussed on page 898, one can set up operator evolution systems similar to symbolic systems (see page 103) that have

essentially the same relationship to operator systems as sequential substitution systems do to multiway systems. (See also page 1172.)

■ **Page 802 · Operator systems.** One can represent the possible values of expressions like *f[f[p, q], p]* by rule numbers analogous to those used for cellular automata. Specifying an operator *f* (taken in general to have *n* arguments with *k* possible values) by giving the rule number *u* for *f[p, q, …]*, the rule number for an expression with variables *vars* can be obtained from

```
With[{m = Length[vars]}, FromDigits[
  Block[{f = Reverse[IntegerDigits[u, k, k^n]][[FromDigits[
      {##}, k] + 1]] &}, Apply[Function[Evaluate[vars], expr],
    Reverse[Array[IntegerDigits[# - 1, k, m] &, k^m]], {1}]], k]]
```

■ **Truth tables.** The method of finding results in logic by enumerating all possible combinations of truth values seems to have been rediscovered many times since antiquity. It began to appear regularly in the late 1800s, and became widely known after its use by Emil Post and Ludwig Wittgenstein in the early 1920s.

■ **Page 803 · Proofs of axiom systems.** One way to prove that an axiom system can reproduce all equivalences for a given operator is to show that its axioms can be used to transform any expression to and from a unique standard form. For then one can start with an expression, convert it to standard form, then convert back to any expression that is equivalent. We saw on page 616 that in ordinary logic there is a unique DNF representation in terms of *And*, *Or* and *Not* for any expression, and in 1921 Emil Post used essentially this to give the first proof that an axiom system like the first one on page 773 can completely reproduce all theorems of logic. A standard form in terms of *Nand* can be constructed essentially by direct translation of DNF; other methods can be used for the various other operators shown. (See also page 1175.)

Given a particular axiom system that one knows reproduces all equivalences for a given operator one can tell whether a new axiom system will also work by seeing whether it can be used to derive the axioms in the original system. But often the derivations needed may be very long—as on page 810. And in fact in 1948 Samuel Linial and Emil Post showed that in general the problem is undecidable. They did this in effect by arguing (much as on page 1169) that any multiway system can be emulated by an axiom system of the form on page 803, then knowing that in general it is undecidable whether a multiway system will ever reach some given result. (Note that if an axiom system does manage to reproduce logic in full then as indicated on page 814 its consequences can always be derived by proofs of limited length, if nothing else by using truth tables.)

Since before 1920 it has been known that one way to disprove the validity of a particular axiom system is to show that with $k > 2$ truth values it allows additional operators (see page 805). (Note that even if it works for all finite k this does not establish its validity.) Another way to do this is to look for invariants that should not be present—seeing if there are features that differ between equivalent expressions, yet are always left the same by transformations in the axiom system. (Examples for logic are axiom systems which never change the size of an expression, or which are of the form *{expr == a}* where *Flatten[expr]* begins or ends with *a*.)

■ **Junctional calculus.** Expressions are equivalent when *Union[Level[expr, {-1}]]* is the same, and this canonical form can be obtained from the axiom system of page 803 by flattening using $(a \circ b) \circ c == a \circ (b \circ c)$, sorting using $a \circ b == b \circ a$, and removing repeats using $a \circ a == a$. The operator can be either *And* or *Or* (8 or 14). With $k = 3$ there are 9 operators that yield the same results:

{13203, 15633, 15663, 16401,
17139, 18063, 19539, 19569, 19599}

With $k = 4$ there are 3944 such operators (see below). No single axiom can reproduce all equivalences, since such an axiom must of the form *expr == a*, yet *expr* cannot contain variables other than *a*, and so cannot for example reproduce $a \circ b = b \circ a$.

■ **Equivalential calculus.** Expressions with variables *vars* are equivalent if they give the same results for

Mod[Map[Count[expr, #, {-1}] &, vars], 2]

With n variables, there are thus 2^n equivalence classes of expressions (compared to 2^{2^n} for ordinary logic). The operator can be either *Xor* or *Equal* (6 or 9). With $k = 3$ there are no operators that yield the same results; with $k = 4$ *{458142180, 1310450865, 2984516430, 3836825115}* work (see below). The shortest axiom system that works up to $k = 2$ is $\{(a \circ b) \circ a = b\}$. With *modus ponens* as the rule of inference, the shortest single-axiom system that works is known to be $\{(a \circ b) \circ ((c \circ b) \circ (a \circ c))\}$. Note that equivalential calculus describes the smallest non-trivial group, and can be viewed as an extremely minimal model of algebra.

■ **Implicational calculus.** With $k = 2$ the operator can be either 2 or 11 (*Implies*), with $k = 3$ *{2694, 9337, 15980}*, and with $k = 4$ any of 16 possibilities. (Operators exist for any k.) No single axiom, at least with up to 7 operators and 4 variables, reproduces all equivalences. With *modus ponens* as the rule of inference, the shortest single-axiom system that works is known to be $\{((a \circ b) \circ c) \circ ((c \circ a) \circ (d \circ a))\}$. Using the method of page 1151 this can be converted to the equational form

$\{((a \circ b) \circ c) \circ ((c \circ a) \circ (d \circ a)) = d \circ d,$
$(a \circ a) \circ b = b, (a \circ b) \circ b = (b \circ a) \circ a\}$

from which the validity of the axiom system in the main text can be established.

■ **Page 803 · Operators on sets.** There is always more than one operator that yields a given collection of equivalences. So for ordinary logic both *Nand* and *Nor* work. And with $k = 4$ any of the 12 operators

```
{1116699, 169585339, 290790239, 459258879,
  1090522958, 1309671358, 1430343258, 1515110058,
  2184380593, 2575151445, 2863760025, 2986292093}
```

also turn out to work. One can see why this happens by considering the analogy between operations in logic and operations on sets. As reflected in their traditional notations—and emphasized by Venn diagrams—*And* (∧), *Or* (∨) and *Not* correspond directly to *Intersection* (∩), *Union* (∪) and *Complement*. If one starts from the single-element set *{1}* then applying *Union*, *Intersection* and *Complement* one always gets either *{}* or *{1}*. And applying *Complement[s, Intersection[a, b]]* to these two elements gives the same results and same equivalences as $a \barwedge b$ applied to *True* and *False*. But if one uses instead *s = {1, 2}* then starts with *{1}* and *{2}* one gets any of *{{}, {1}, {2}, {1, 2}}* and in general with *s = Range[n]* one gets any of the 2^n elements in the powerset

```
Distribute[Map[{{}, {#}} &, s], List, List, List, Join]
```

But applying *Complement[s, Intersection[a, b]]* to these elements still always produces the same equivalences as with $a \barwedge b$. Yet now $k = 2^n$. And so one therefore has a representation of Boolean algebra of size 2^n. For ordinary logic based on *Nand* it turns out that there are no other finite representations (though there are other infinite ones). But if one works, say, with *Implies* then there are for example representations of size 3 (see above). And the reason for this is that with *s = {1, 2}* the function *Union[Complement[s, a], b]* corresponding to $a \Rightarrow b$ only ever gets to the 3 elements *{{1}, {2}, {1, 2}}*. Indeed, in general with operators *Implies*, *And* and *Or* one gets to $2^n - 1$ elements, while with operators *Xor* and *Equal* one gets to *2^(2 Floor[n/2])* elements.

(One might think that one could force there only ever to be two elements by adding an axiom like $a == b \vee b == c \vee c == a$. But all this actually does is to force there to be only two objects analogous to *True* and *False*.)

■ **Page 805 · Implementation.** Given an axiom system in the form *{f[a, f[a, a]] = a, f[a, b] = f[b, a]}* one can find rule numbers for the operators *f[x, y]* with *k* values for each variable that are consistent with the axiom system by using

```
Module[{c, v}, c = Apply[Function,
  {v = Union[Level[axioms, {-1}]], Apply[And, axioms]}];
 Select[Range[0, k^2 - 1], With[{u = IntegerDigits[#, k, k^2]},
   Block[{f}, f[x_, y_] := u[[-1 - k x - y]];
    Array[c, Table[k, {Length[v]}], 0, And]]] &]]
```

For $k = 4$ this involves checking nearly 16^4 or 4 billion cases, though many of these can often be avoided, for example by using analogs of the so-called Davis-Putnam rules. (In searching for an axiom system for a given operator it is in practice often convenient first to test whether each candidate axiom holds for the operator one wants.)

■ **Page 805 · Properties.** There are k^{k^2} possible forms for binary operators with *k* possible values for each argument. There is always at least some operator that satisfies the constraints of any given axiom system—though in a case like $a = b$ it has $k = 1$. Of the 274,499 axiom systems of the form *{… = a}* where … involves ∘ up to 6 times, 32,004 allow only operators *{6, 9}*, while 964 allow only *{1, 7}*. The only cases of 2 or less operators that appear with $k = 2$ are *{{}, {10}, {12}, {1, 7}, {3, 12}, {5, 10}, {6, 9}, {10, 12}}*. (See page 1174.)

■ **Page 806 · Algebraic systems.** Operator systems can be viewed as algebraic systems of the kind studied in universal algebra (see page 1150). With a single two-argument operator (such as ∘) what one has is in general known as a groupoid (though this term means something different in topology and category theory); with two such operators a ringoid. Given a particular algebraic system, it is sometimes possible—as we saw on page 773—to reduce the number of operators it involves. But the number of systems that have traditionally been studied in mathematics and that are known to require only one 2-argument operator are fairly limited. In addition to basic logic, semigroups and groups, there are essentially only the rather obscure examples of semilattices, with axioms *{a∘(b∘c) = (a∘b)∘c, a∘b = b∘a, a∘a = a}*, central groupoids, with axioms *{(b∘a)∘(a∘c) = b}*, and squags (quasigroup representations of Steiner triple systems), with axioms *{a∘b = b∘a, a∘a = a, a∘(a∘b) = b}* or equivalently *{a∘((b∘(b∘(((c∘c)∘d)∘c)))∘a) = d}*. (Ordinary quasigroups are defined by *{a∘c = b, d∘a = b}* with *c*, *d* unique for given *a*, *b*—so that their table is a Latin square; their axioms can be set up with 3 operators as *{a\a∘b = b, a∘b/b = a, a∘(a\b) = b, (a/b)∘b = a}*.)

Pages 773 and 774 indicate that most axiom systems in mathematics involve operators with at most 2 arguments (there are exceptions in geometry). (Constants such as *1* or ∅ can be viewed as 0-argument operators.) One can nevertheless generalize say to polyadic groups, with 3-argument composition and analogs of associativity such as

```
f[f[a, b, c], d, e] = f[a, f[b, c, d], e] = f[a, b, f[c, d, e]]
```

Another example is the cellular automaton axiom system of page 794; see also page 886. (A perhaps important

generalization is to have expressions that are arbitrary networks rather than just trees.)

■ **Symbolic systems.** By introducing constants (0-argument operators) and interpreting ∘ as function application one can turn any symbolic system such as $e[x][y] \to x[x[y]]$ from page 103 into an algebraic system such as $(e \circ a) \circ b = a \circ (a \circ b)$. Doing this for the combinator system from page 711 yields the so-called combinatory algebra $\{((s \circ a) \circ b) \circ c = (a \circ c) \circ (b \circ c), (k \circ a) \circ b = a\}$.

■ **Page 806 · Groups and semigroups.** With k possible values for each variable, the forms of operators allowed by axiom systems for group theory and semigroup theory correspond to multiplication tables for groups and semigroups with k elements. Note that the first group that is not commutative (Abelian) is the group S3 with $k = 6$ elements. The total number of commutative groups with k elements is just

```
Apply[Times,
  Map[PartitionsP[Last[#]] &, FactorInteger[k]]]
```

(Relabelling of elements makes the number of possible operator forms up to $k!$ times larger.) (See also pages 945, 1153 and 1173.)

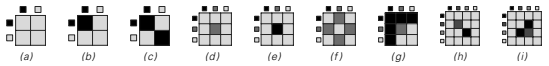
■ **Forcing of operators.** Given a particular set of forms for operators one can ask whether an axiom system can be found that will allow these but no others. As discussed in the note on operators on sets on page 1171 some straightforwardly equivalent forms will always be allowed. And unless one limits the number of elements k it is in general undecidable whether a given axiom system will allow no more than a given set of forms. But even with fixed k it is also often not possible to force a particular set of forms. And as an example of this one can consider commutative group theory. The basic axioms for this allow forms of operators corresponding to multiplication tables for all possible commutative groups (see note above). So to force particular forms of operators would require setting up axioms satisfied only by specific commutative groups. But it turns out that given the basic axioms for commutative group theory any non-trivial set of additional axioms can always be reduced to a single axiom of the form $a^n == 1$ (where exponentiation is repeated application of ∘). Yet even given a particular number of elements k, there can be several distinct groups satisfying $a^n == 1$ for a given exponent n. (The groups can be written as products of cyclic ones whose orders correspond to the possible factors of n.) (Something similar is also known in principle to be true for general groups, though the hierarchy of axioms in this case is much more complicated.)

■ **Model theory.** In model theory each form of operator that satisfies the constraints of a given axiom system is called a model of that axiom system. If there is only one inequivalent model the axiom system is said to be categorical—a notion discussed for example by Richard Dedekind in 1887. The Löwenheim-Skolem theorem from 1915 implies that any axiom system must always have a countable model. (For an operator system such a model can have elements which are simply equivalence classes of expressions equal according to the axioms.) So this means that even if one tries to set up an axiom system to describe an uncountable set—such as real numbers—there will inevitably always be extra countable models. Any axiom system that is incomplete must always allow more than one model. The model intended when the axiom system was originally set up is usually called the standard model; others are called non-standard. In arithmetic non-standard models are obscure, as discussed on page 1169. In analysis, however, Abraham Robinson pointed out in 1960 that one can potentially have useful non-standard models, in which there are explicit infinitesimals—much along the lines suggested in the original work on calculus in the late 1600s.

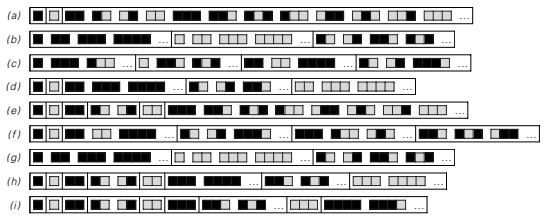
■ **Pure equational logic.** Proofs in operator systems always rely on certain underlying rules about equality, such as the equivalence of $u == v$ and $v == u$, and of $u == v$ and $u == v /. a \to b$. And as Garrett Birkhoff showed in 1935, any equivalence between expressions that holds for all possible forms of operator must have a finite proof using just these rules. (This is the analog of Gödel's Completeness Theorem from page 1152 for pure predicate logic.) But as soon as one introduces actual axioms that constrain the operators this is no longer true—and in general it can be undecidable whether or not a particular equivalence holds.

■ **Multiway systems.** One can use ideas from operator systems to work out equivalences in multiway systems (compare page 1169). One can think of concatenation of strings as being an operator, in terms of which a string like `"ABB"` can be written `f[f[a, b], b]`. (The arguments to ∘ should strictly be distinct constants, but no equivalences are lost by allowing them to be general variables.) Assuming that the rules for a multiway system come in pairs $p \to q$, $q \to p$, like $\text{"AB"} \to \text{"AAA"}$, $\text{"AAA"} \to \text{"AB"}$, these can be written as statements about operators, like `f[a, b] == f[f[a, a], a]`. The basic properties of concatenation then also imply that `f[f[a, b], c] == f[a, f[b, c]]`. And this means that the possible forms for the operator ∘ correspond to possible semigroups. Given a particular such semigroup satisfying axioms derived from a multiway system, one can see whether the operator representations of particular strings are equal—and if they are not, then it follows that the strings can never be reached from each other through evolution of the multiway system. (Such operator representations are a rough analog for multiway systems of

truth tables.) As an example, with the multiway system *"AB" ↔ "BA"* some possible forms of operators are shown below. (In this case these are the commutative semigroups. With $k = 2$, elements 6 out of the total of 8 possible semigroups appear; with $k = 3$, 63 out of 113, and with $k = 4$, 1140 out of 3492—all as shown on page 805.) (See also page 952.)



Taking ∘ to be each of these operators, one can work out a representation for any given string like *"ABAA"* by for example constructing the expression *f[f[f[a, b], a], a]* and finding its value for each of the k^2 possible pairs of values of a and b. Then for each successive operator, the sets of strings where the arrays of values are the same are as shown below.



Ultimately the sets of strings equivalent under the multiway system are exactly those containing particular numbers of black and white elements. But as the pictures above suggest, only some of the distinctions between sets of strings are ever captured when any specific form for the operator is used.

Just as for operator systems, any bidirectional multiway system will allow a certain set of operators. (When there are multiple rules in the multiway system, tighter constraints are obtained by combining them with *And*.) And the pattern of results for simple multiway systems is roughly similar to those on page 805 for operator systems—although, for example, the associativity of concatenation makes it impossible for example to get the operators for *Nand* and basic logic.

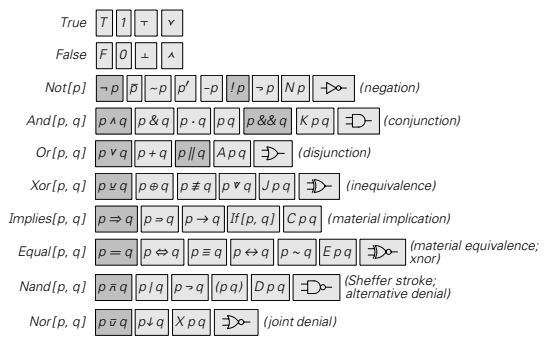
▪ **Page 806 · Logic in languages.** Human languages always seem to have single words for AND, OR and NOT. A few have distinct words for OR and XOR: examples are Latin with *vel* and *aut* and Finnish with *vai* and *tai*. NOR is somewhat rare, though Dutch has *noch* and Old English *ne*. (Modern English has only the compound form *neither ... nor*.) But remarkably enough it appears that no ordinary language has a single word for NAND. The reason is not clear. Most people seem to find it difficult to think in terms of NAND (NAND is for example not associative, but then neither is NOR). And NAND on the face of it rarely seems useful in everyday situations. But perhaps these are just reflections of the historical fact that NAND has never been familiar from ordinary languages.

Essentially all computer languages support AND, OR and NOT as ways to combine logical statements; many support AND, OR and XOR as bitwise operations. Circuit design languages like Verilog and VHDL also support NAND, NOR and XNOR. (NAND is the operation easiest to implement with CMOS FETs—the transistors used in most current chips; it was also implemented by pentode vacuum tubes.) Circuit designers sometimes use the linguistic construct "*p* nand *q*".

The Laws of Form presented by George Spencer Brown in 1969 introduce a compact symbolic notation for NAND with any number of arguments and in effect try to develop a way of discussing NAND and reasoning directly in terms of it. (The axioms normally used are essentially the Sheffer ones from page 773.)

▪ **Page 806 · Properties.** Page 813 lists theorems satisfied by each function. *{0, 1, 6, 7, 8, 9, 14, 15}* are commutative (orderless) so that $a \circ b = b \circ a$, while *{0, 6, 8, 9, 10, 12, 14, 15}* are associative (flat), so that $a \circ (b \circ c) = (a \circ b) \circ c$. (Compare page 886.)

▪ **Notations.** Among those in current use are (highlighted ones are supported directly in *Mathematica*):



The grouping of terms is normally inferred from precedence of operators (typically ordered $==, \neg, \barwedge, \wedge, \veebar, \barvee, \vee, \Rightarrow$), or explicitly indicated by parentheses, function brackets, or sometimes nested underbars or dots. So-called Polish notation given second-to-last above avoids all explicit delimiters (see page 896).

▪ **Page 807 · Universal logical functions.** The fact that combinations of *Nand* or *Nor* are adequate to reproduce any logical function was noted by Charles Peirce around 1880, and became widely known after the work of Henry Sheffer in 1913. (See also page 1096.) *Nand* and *Nor* are the only 2-input functions universal in this sense. (*{Equal}* can for example

reproduce only functions *{9, 10, 12, 15}*, *{Implies}* only functions *{10, 11, 12, 13, 14, 15}*, and *{Equal, Implies}* only functions *{8, 9, 10, 11, 12, 13, 14, 15}*.) For 3-input functions, corresponding to elementary cellular automaton rules, 56 of the 256 possibilities turn out to be universal. Of these, 6 are straightforward generalizations of *Nand* and *Nor*. Other universal functions include rules 1, 45 and 202 (*If[a == 1, b, c]*), but not 30, 60 or 110. For large n roughly 1/4 of all n-input functions are universal. (See also page 1175.)

■ **Page 808 · Searching for logic.** For axiom systems of the form *{… = a}* one finds:

	2 variables					3 variables				
number of ∘	2	3	4	5	6	2	3	4	5	6
total systems	4	16	80	448	2688	54	405	3402	30618	288684
allow $\bar{\pi}$	0	5	44	168	1532	0	9	124	744	8764
allow only $\bar{\pi}$ etc. for k=2	0	0	2	12	76	0	0	12	84	868
allow only $\bar{\pi}$ etc. for k≤3	0	0	0	0	0	0	0	8	16	296
allow only $\bar{\pi}$ etc. for k≤4	0	0	0	0	0	0	0	0	0	100

{((b∘b)∘a)∘(a∘b) = a} allows the *k = 3* operator 15552 for which the NAND theorem *(p∘p)∘q = (p∘q)∘q* is not true. *{((((b∘a)∘c)∘a)∘(a∘c) = a}* allows the *k = 4* operator 95356335 for which even *p∘q = q∘p* is not true. Of the 100 cases that remain when *k = 4*, the 25 inequivalent under renaming of variables and reversing arguments of ∘ are

{(b∘(b∘(a∘a)))∘(a∘(b∘c)),
(b∘(b∘(a∘a)))∘(a∘(c∘b)), (b∘(b∘(a∘b)))∘(a∘(b∘c)),
(b∘(b∘(a∘b)))∘(a∘(c∘b)), (b∘(b∘(a∘c)))∘(a∘(c∘b)),
(b∘(b∘(b∘a)))∘(a∘(b∘c)), (b∘(b∘(b∘a)))∘(a∘(c∘b)),
(b∘(b∘(c∘a)))∘(a∘(b∘c)), (b∘((a∘b)∘b))∘(a∘(b∘c)),
(b∘((a∘b)∘b))∘(a∘(c∘b)), (b∘((a∘c)∘b))∘(a∘(c∘b)),
((b∘c)∘a)∘(b∘(b∘(a∘b))), ((b∘c)∘a)∘(b∘(b∘(a∘c))),
((b∘c)∘a)∘(b∘((a∘a)∘b)), ((b∘c)∘a)∘(b∘((a∘b)∘b)),
((b∘c)∘a)∘(b∘((a∘c)∘b)), ((b∘c)∘a)∘(b∘((b∘a)∘b)),
((b∘c)∘a)∘(b∘((c∘a)∘b)), ((b∘c)∘a)∘(c∘(c∘(a∘b))),
((b∘c)∘a)∘(c∘(c∘(a∘c))), ((b∘c)∘a)∘(c∘((a∘a)∘c)),
((b∘c)∘a)∘(c∘((a∘b)∘c)), ((b∘c)∘a)∘(c∘((a∘c)∘c)),
((b∘c)∘a)∘(c∘((b∘a)∘c)), ((b∘c)∘a)∘(c∘((c∘a)∘c))}

Of these I was able in 2000—using automated theorem proving—to show that the ones given as (g) and (h) in the main text are indeed axiom systems for logic. (My proof essentially as found by Waldmeister is given on page 810.)

If one adds *a∘b = b∘a* to any of the other 23 axioms above then in all cases the resulting axiom system can be shown to reproduce logic. But from any of the 23 axioms on their own I have never managed to derive *p∘q = q∘p*. Indeed, it seems difficult to derive much at all from them—though for example I have found a fairly short proof of *(p∘p)∘(p∘q) = p* from *{(b∘(b∘(b∘a)))∘(a∘(b∘c)) = a}*.

It turns out that the first of the 25 axioms allows the *k = 6* operator *1885760537655023865453442036* and so cannot be logic. Axioms 3, 19 and 23 allow similar operators, leaving 19 systems as candidate axioms for logic.

It has been known since the 1940s that any axiom system for logic must have at least one axiom that involves more than 2 variables. The results above now show that 3 variables suffice. And adding more variables does not seem to help. The smallest axiom systems with more than 3 variables that work up to *k = 2* are of the form *{((((b∘c)∘d)∘a)∘(a∘d) = a}*. All turn out also to work at *k = 3*, but fail at *k = 4*. And with 6 NANDs (as in (g) and (h)) no system of the form *{… = a}* works even up to *k = 4*.

For axiom systems of the form *{… = a, a∘b = b∘a}*:

	2 variables					3 variables				
number of ∘	4	5	6	7	8	4	5	6	7	8
total systems	4	16	80	448	2688	54	405	3402	30618	288684
allow $\bar{\pi}$	0	5	44	168	1532	0	9	124	744	8764
allow only $\bar{\pi}$ etc. for k=2	0	4	20	160	748	0	8	80	736	6248
allow only $\bar{\pi}$ etc. for k≤3	0	0	0	64	16	0	0	32	416	2752
allow only $\bar{\pi}$ etc. for k≤4	0	0	0	48	16	0	0	32	384	2368

With 2 variables the inequivalent cases that remain are

{(a∘b)∘(a∘(b∘(a∘b))),
(a∘b)∘(a∘(b∘(b∘b))), (a∘(b∘b))∘(a∘(b∘(b∘b))))}

but all of these allow the *k = 6* operator

188576053765512542973848088 4

and so cannot correspond to basic logic. With 3 variables, all 32 cases with 6 NANDs are equivalent to *(a∘b)∘(a∘(b∘c))*, which is axiom system (f) in the main text. With 7 NANDs there are 8 inequivalent cases:

{(a∘a)∘(b∘(b∘(a∘c))), (a∘b)∘(a∘(b∘(a∘b))), (a∘b)∘(a∘(b∘(a∘c))),
(a∘b)∘(a∘(b∘(b∘b))), (a∘b)∘(a∘(b∘(b∘c))),
(a∘b)∘(a∘(b∘(c∘c))), (a∘b)∘(a∘(c∘(a∘c))), (a∘b)∘(a∘(c∘(c∘c))))}

and of these at least 5 and 6 can readily be proved to be axioms for logic.

Any axiom system must consist of equivalences valid for the operator it describes. But the fact that there are fairly few short such equivalences for *Nand* (see page 818) implies that there can be no axiom system for *Nand* with 6 or less NANDs except the ones discussed above.

■ **Two-operator logic.** If one allows two operators then one can get standard logic if one of these operators is forced to be *Not* and the other is forced to be *And*, *Or* or *Implies*—or in fact any of operators 1, 2, 4, 7, 8, 11, 13, 14 from page 806.

A simple example that allows *Not* and either *And* or *Or* is the Robbins axiom system from page 773. Given the first two axioms (commutativity and associativity) it turns out that no shorter third axiom will work in this case (though ones such as *f[g[f[a, g[f[a, b]]]], g[g[b]]] = b* of the same size do work).

Much as in the single-operator case, to reproduce logic two pairs of operators must be allowed for *k = 2*, none for *k = 3*, 12 for *k = 4*, and so on. Among single axioms, the shortest that works up to *k = 2* is *(¬ (¬ (¬ b ∨ a) ∨ ¬ (a ∨ b))) = a*. The shortest that

works up to $k = 3$ is $(\neg(\neg(a \vee b) \vee \neg b) \vee \neg(\neg a \vee a)) = b$. It is known, however, that at least 3 variables must appear in order to reproduce logic, and an example of a single axiom with 4 variables that has been found recently to work is $\{(\neg(\neg(c \vee b) \vee \neg a) \vee \neg(\neg(\neg d \vee d) \vee \neg a \vee c)) = a\}$.

■ **Page 808 · History.** (See page 1151.) (c) was found by Henry Sheffer in 1913; (e) by Carew Meredith in 1967. Until this book, very little work appears to have been done on finding short axioms for logic in terms of *Nand*. Around 1949 Meredith found the axiom system

$$\{(a \circ (b \circ c)) \circ (a \circ (b \circ c)) = ((c \circ a) \circ a) \circ ((b \circ a) \circ a),\ (a \circ a) \circ (b \circ a) = a\}$$

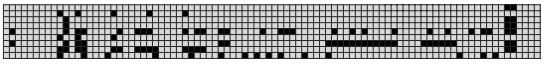
In 1967 George Spencer Brown found (see page 1173)

$$\{(a \circ a) \circ ((b \circ b) \circ b) = a,\ a \circ (b \circ c) = (((c \circ c) \circ a) \circ ((b \circ b) \circ a)) \circ (((c \circ c) \circ a) \circ ((b \circ b) \circ a))\}$$

and in 1969 Meredith also gave the system

$$\{a \circ (b \circ (a \circ c)) = a \circ (b \circ (b \circ c)),\ (a \circ a) \circ (b \circ a) = a,\ a \circ b = b \circ a\}$$

■ **Page 812 · Theorem distributions.** The picture below shows which of the possible theorems from page 812 hold for each of the numbered standard mathematical theories from page 805. The theorem close to the right-hand end valid in many cases is $(p \circ p) \circ p = p \circ (p \circ p)$. The lack of regularity in this picture can be viewed as a sign that it is difficult to tell which theorems hold, and thus in effect to do mathematics.

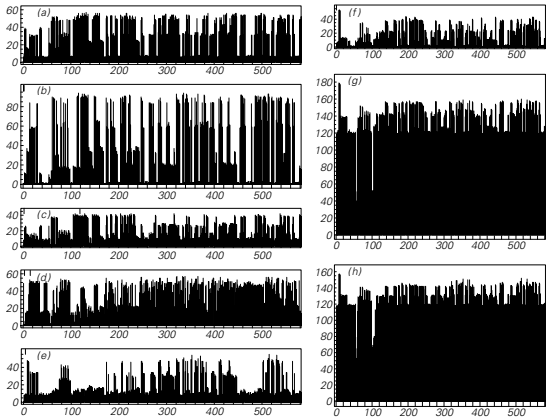


■ **Page 814 · Multivalued logic.** As noted by Jan Łukasiewicz and Emil Post in the early 1920s, it is possible to generalize ordinary logic to allow k values *Range[0, 1, 1/(k - 1)]*, say with 0 being *False*, and 1 being *True*. Standard operations in logic can be generalized as *Not[a_] = 1 - a*, *And[a_, b_] = Min[a, b]*, *Or[a_, b_] = Max[a, b]*, *Xor[a_, b_] = Abs[a - b]*, *Equal[a_, b_] = 1 - Abs[a - b]*, *Implies[a_, b_] = 1 - UnitStep[a - b] (a - b)*. An alternative generalization for *Not* is *Not[a_] := Mod[(k - 1) a + 1, k]/(k - 1)*. The function *Nand[a_, b_] := Not[And[a, b]]* used in the main text turns out to be universal for any k. Axiom systems can be set up for multivalued logic, but they are presumably more complicated than for ordinary $k = 2$ logic. (Compare page 1171.)

The idea of intermediate truth values has been discussed intermittently ever since antiquity. Often—as in the work of George Boole in 1847—a continuum of values between 0 and 1 are taken to represent probabilities of events, and this is the basis for the field of fuzzy logic popular since the 1980s.

■ **Page 814 · Proof lengths in logic.** As discussed on page 1170 equivalence between expressions can always be proved by transforming to and from canonical form. But with n variables a DNF-type canonical form can be of size 2^n—and can take up to at least 2^n proof steps to reach. And indeed if logic proofs could in general be done in a number of steps that increases only like a polynomial in n this would mean that the NP-complete problem of satisfiability could also be solved in this number of steps, which seems very unlikely (see page 768).

In practice it is usually extremely difficult to find the absolute shortest proof of a given logic theorem—and the exact length will depend on what axiom system is used, and what kinds of steps are allowed. In fact, as mentioned on page 1155, if one does not allow lemmas some proofs perhaps have to become exponentially longer. The picture below shows in each of the axiom systems from page 808 the lengths of the shortest proofs found by a version of Waldmeister (see page 1158) for all 582 equivalences (see page 818) that involve two variables and up to 3 NANDs on either side.



The longest of these are respectively $\{57, 94, 42, 57, 55, 53, 179, 157\}$ and occur for theorems

$$\{(((a \barwedge a) \barwedge b) \barwedge b) = (((a \barwedge b) \barwedge a) \barwedge a),\ (a \barwedge (a \barwedge (a \barwedge a))) = (a \barwedge ((a \barwedge b) \barwedge b)),\ (((a \barwedge a) \barwedge a) \barwedge a) = (((a \barwedge a) \barwedge b) \barwedge a),\ (((a \barwedge a) \barwedge b) \barwedge b) = (((a \barwedge b) \barwedge a) \barwedge a),\ (a \barwedge ((b \barwedge b) \barwedge a)) = (b \barwedge ((a \barwedge a) \barwedge b)),\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b),\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b),\ ((a \barwedge a) \barwedge a) = ((b \barwedge b) \barwedge b)\}$$

Note that for systems that do not already have it as an axiom, most theorems use the lemma $(a \barwedge b) = (b \barwedge a)$ which takes respectively $\{6, 1, 8, 49, 8, 1, 119, 118\}$ steps to prove.

■ **Page 818 · NAND theorems.** The total number of expressions with n NANDs and s variables is: *Binomial[2 n, n] s^{n+1}/(n + 1)* (see page 897). With $s = 2$ and n from 0 to 7 the number of these *True* for all values of variables is $\{0, 0, 4, 0, 80, 108, 2592, 7296\}$, with the first few distinct ones being (see page 781)

$$\{(p \barwedge p) \barwedge p,\ (((p \barwedge p) \barwedge p) \barwedge p) \barwedge p,\ (((p \barwedge p) \barwedge p) \barwedge q) \barwedge q\}$$

The number of unequal expressions obtained is *{2, 3, 3, 7, 10, 15, 12, 16}* (compare page 1096), with the first few distinct ones being

{p, p ⊼ p, p ⊼ q, (p ⊼ p) ⊼ p, (p ⊼ q) ⊼ p, (p ⊼ p) ⊼ q}

Most of the axioms from page 808 are too long to appear early in the list of theorems. But those of system (d) appear at positions *{3, 15, 568}* and those of (e) at *{855, 4}*.

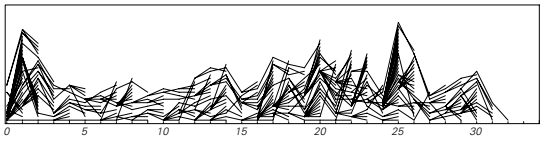
(See also page 1096.)

▪ **Page 819 · Finite axiomatizability.** It is known that the axiom systems (such as Peano arithmetic and set theory) given with axiom schemas on pages 773 and 774 can be set up only with an infinite number of individual axioms. But because such axioms can be described by schemas they must all have similar forms, so that even though the definition in the main text suggests that each corresponds to an interesting theorem these theorems are not in a sense independently interesting. (Note that for example the theory of specifically finite groups cannot be set up with a finite number even of schemas—or with any finite procedure for checking whether a given candidate axiom should be included.)

▪ **Page 820 · Empirical metamathematics.** One can imagine a network representing some field of mathematics, with nodes corresponding to theorems and connections corresponding to proofs, gradually getting filled in as the field develops. Typical rough characterizations of mathematical results—independent of their detailed history—might then end up being something like the following:

- lemma: short theorem appearing at an intermediate stage in a proof
- corollary: theorem connected by a short proof to an existing theorem
- easy theorem: theorem having a short proof
- difficult theorem: theorem having a long proof
- elegant theorem: theorem whose statement is short and somewhat unique
- interesting theorem (see page 817): theorem that cannot readily be deduced from earlier theorems, but is well connected
- boring theorem: theorem for which there are many others very much like it
- useful theorem: theorem that leads to many new ones
- powerful theorem: theorem that substantially reduces the lengths of proofs needed for many others
- surprising theorem: theorem that appears in an otherwise sparse part of the network of theorems
- deep theorem: theorem that connects components of the network of theorems that are otherwise far away
- important theorem: theorem that allows a broad new area of the network to be reached.

The picture below shows the network of theorems associated with Euclid's *Elements*. Each stated theorem is represented by a node connected to the theorems used in its stated proof. (Only the shortest connection from each theorem is shown explicitly.) The axioms (postulates and common notions) are given in the first column on the left, and successive columns then show theorems with progressively longer proofs. (Explicit annotations giving theorems used in proofs were apparently added to editions of Euclid only in the past few centuries; the picture below extends the usual annotations in a few cases.) The theorem with the longest proof is the one that states that there are only five Platonic solids.



▪ **Speedups in other systems.** Multiway systems are almost unique in being able to be sped up just by adding "results" already derived in the multiway system. In other systems, there is no such direct way to insert such results into the rules for the system.

▪ **Character of mathematics.** Since at least the early 1900s several major schools of thought have existed:

- *Formalism* (e.g. David Hilbert): Mathematics studies formal rules that have no intrinsic meaning, but are relevant because of their applications or history.
- *Platonism* (e.g. Kurt Gödel): Mathematics involves trying to discover the properties of a world of ideal mathematical forms, of which we in effect perceive only shadows.
- *Logicism* (e.g. Gottlob Frege, Bertrand Russell): Mathematics is an elaborate application of logic, which is itself fundamental.
- *Intuitionism* (e.g. Luitzen Brouwer): Mathematics is a precise way of handling ideas that are intuitive to the human mind.

The results in this book establish a new point of view somewhere between all of these.

▪ **Invention versus discovery in mathematics.** One generally considers things invented if they are created in a somewhat arbitrary way, and discovered if they are identified by what

seems like a more inexorable process. The results of this section thus strongly suggest that the basic directions taken by mathematics as currently practiced should mostly be considered invented, not discovered. The new kind of science that I describe in this book, however, tends to suggest forms of mathematics that involve discovery rather than invention.

▪ **Ordering of constructs.** One can deduce some kind of ordering among standard mathematical constructs by seeing how difficult they are to implement in various systems—such as cellular automata, Turing machines and Diophantine equations. My experience has usually been that addition is easiest, followed by multiplication, powers, Fibonacci numbers, perfect numbers and then primes. And perhaps this is similar to the order in which these constructs appeared in the early history of mathematics. (Compare page 640.)

▪ **Mathematics and the brain.** A possible reason for some constructs to be more common in mathematics is that they are somehow easier for human brains to manipulate. Typical human experience makes small positive integers and simple shapes familiar—so that all human brains are at least well adapted to such constructs. Yet of the limited set of people exposed to higher mathematics, different ones often seem to think in bizarrely different ways. Some think symbolically, presumably applying linguistic capabilities to algebraic or other representations. Some think more visually, using mechanical experience or visual memory. Others seem to think in terms of abstract patterns, perhaps sometimes with implicit analogies to musical harmony. And still others—including some of the purest mathematicians—seem to think directly in terms of constraints, perhaps using some kind of abstraction of everyday geometrical reasoning.

In the history of mathematics there are many concepts that seem to have taken surprisingly long to emerge. And sometimes these are ones that people still find hard to grasp. But they often later seem quite simple and obvious—as with many examples in this book.

It is sometimes thought that people understand concepts in mathematics most easily if they are presented in the order in which they arose historically. But for example the basic notion of programmability seems at some level quite easy even for young children to grasp—even though historically it appeared only recently.

In designing *Mathematica* one of my challenges was to use constructs that are at least ultimately easy for people to understand. Important criteria for this in my experience include specifying processes directly rather than through constraints, the explicitness in the representation of input and output, and the existence of small, memorable, examples. Typically it seems more difficult for people to understand processes in which larger numbers of different things happen in parallel. (Notably, *FoldList* normally seems more difficult to understand than *NestList*.) Tree structures such as *Mathematica* expressions are fairly easy to understand. But I have never found a way to make general networks similarly easy, and I am beginning to suspect that they may be fundamentally difficult for brains to handle.

▪ **Page 821 · Frameworks.** Symbolic integration was in the past done by a collection of ad hoc methods like substitution, partial fractions, integration by parts, and parametric differentiation. But in *Mathematica Integrate* is now almost completely systematic, being based on structure theorems for finding general forms of integrals, and on general representations in terms of *MeijerG* and other functions. (In recognizing, for example, whether an expression involving a parameter can have a pole undecidable questions can in principle come up, but they seem rare in practice.) Proofs are essentially always still done in an ad hoc way—with a few minor frameworks like enumeration of cases, induction, and proof by contradiction (reductio ad absurdum) occasionally being used. (More detailed frameworks are used in specific areas; an example are ϵ-δ arguments in calculus.) But although still almost unknown in mainstream mathematics, methods from automated theorem proving (see page 1157) are beginning to allow proofs of many statements that can be formulated in terms of operator systems to be found in a largely systematic way (e.g. page 810). (In the case of Euclidean geometry—which is a complete axiom system—algebraic methods have allowed complete systematization.) In general, the more systematic the proofs in a particular area become, the less relevant they will typically seem compared to the theorems that they establish as true.

Intelligence in the Universe

▪ **Page 822 · Animism.** Attributing abstract human qualities such as intelligence to systems in nature is a central part of the idea of animism, discussed on page 1195.

▪ **Page 822 · The weather.** Almost all the intricate variations of atmospheric temperature, pressure, velocity and humidity that define the weather we see are in the end determined by fairly simple underlying rules for fluid behavior. (Details of phase changes in water are also important, as are features of topography, ocean currents, solar radiation levels and presumably land use.) Our everyday personal attempts to predict the weather tend to be based just on looking at local conditions and then recalling what happened when we saw these conditions before. But ever since the mid-1800s

synoptic weather maps of large areas have been available that summarize conditions in terms of features like fronts and cyclones. And predictions made by looking at simple trends in these features tend at least in some situations to work fairly well. Starting in the 1940s more systematic efforts to predict weather were made by using computers to run approximations to fluid equations. The approximations have improved as larger computers have become available. But even though millions of discrete samples are now used, each one typically still represents something much larger then for example a single cloud. Yet ever since the 1970s, the approach has had at least some success in making predictions up to several days in advance. But although there has been gradual improvement it is usually assumed that—like in the Lorenz equations—the phenomenon of chaos must make forecasts that are based on even slightly different initial measurements eventually diverge exponentially (see page 972). Almost certainly this does indeed happen in at least some critical situations. But it seems that over most of a typical weather map there is no such sensitivity—so that in the end the difficulties of weather prediction are probably much more a result of computational irreducibility and of the sophisticated kinds of computations that the Principle of Computational Equivalence implies should often occur even in simple fluids.

▪ **Page 822 · Defining intelligence.** The problem of defining intelligence independent of specific education and culture has been considered important for human intelligence testing since the beginning of the 1900s. Charles Spearman suggested in 1904 that there might be a general intelligence factor (usually called g) associated with all intellectual tasks. Its nature was never very clear, but it was thought that its value could be inferred from performance on puzzles involving numbers, words and pictures. By the 1980s, however, there was increasing emphasis on the idea that different types of human tasks require different types of intelligence. But throughout the 1900s psychologists occasionally tried to give general definitions of intelligence—initially usually in terms of learning or problem-solving capabilities; later more often in terms of adaptation to complex environments.

Particularly starting at the end of the 1800s there was great interest in whether animals other than humans could be considered intelligent. The most common general criterion used was the ability to show behavior based on conceptual or abstract thinking rather than just simple instincts. More specific criteria also included ability to use tools, plan actions, use language, solve logical problems and do arithmetic. But by the mid-1900s it became increasingly clear that it was very difficult to interpret actual observations—and that unrecognized cues could for example often account for the behavior seen.

When the field of artificial intelligence began in the mid-1900s there was some discussion of appropriate definitions of intelligence (see page 1099). Most focused on mathematical or other problem solving, though some—such as the Turing test—emphasized everyday conversation with humans.

▪ **Page 823 · Mimesis.** The notion of inanimate analogs of memory—such as impressions in wax—was discussed for example by Plato in antiquity.

▪ **Page 823 · Defining life.** Greek philosophers such as Aristotle defined life by the presence of some form of soul, and the idea that there must be a single unique feature associated with life has always remained popular. In the 1800s the notion of a "life force" was discussed—and thought to be associated perhaps with chemical properties of protoplasm, and perhaps with electricity. The discovery from the mid-1800s to the mid-1900s of all sorts of elaborate chemical processes in living systems led biologists often to view life as defined by its ability to maintain fixed overall structure while achieving chemical functions such as metabolism. When the Second Law of Thermodynamics was formulated in the mid-1800s living systems were usually explicitly excluded (see page 1021), and by the 1930s physicists often considered local entropy decrease a defining feature of life. Among geneticists and soon mathematicians self-reproduction was usually viewed as the defining feature of life, and following the discovery of the structure of DNA in 1953 it came to be widely believed that the presence of self-replicating elements must be fundamental to life. But the recognition that just copying information is fairly easy led in the 1960s to definitions of life based on the large amounts of information encoded in its genetic material, and later to ones based on the apparent difficulty of deriving this information (see page 1069). And perhaps in part reacting to my discoveries about cellular automata it became popular in the 1980s to mention adaptation and essential interdependence of large numbers of different kinds of parts as further necessary characteristics of life. Yet in the end every single general definition that has been given both includes systems that are not normally considered alive, and excludes ones that are. (Self-reproduction, for example, suggests that flames are alive, but mules are not.)

One of the features that defines life on Earth is the presence of DNA, or at least RNA. But as one looks at smaller molecules they become less specific to living systems. It is sometimes thought significant that living systems perpetuate the use of only one chirality of molecules, but actually this

can quite easily be achieved by various forms of non-chemical input without life.

The Viking spacecraft that landed on Mars in the 1970s tried specific tests for life on soil samples—essentially whether gases were generated when nutrients were added, whether this behavior changed if the samples were first heated, and whether molecules common in terrestrial life were present. The tests gave confusing results, presumably having to do not with life, but rather with details of martian soil chemistry

▪ **Origin of life.** Fossil traces of living cells have been found going back more than 3.8 billion years—to perhaps as little as 700 million years after the formation of the Earth. There were presumably simpler forms of life that preceded the advent of recognizable cells, and even if life arose more than once it is unlikely that evidence of this would remain. (One sees many branches in the fossil record—such as organisms with dominant symmetries other than fivefold—but all seem to have the same ancestry.)

From antiquity until the 1700s it was widely believed that smaller living organisms arise spontaneously in substances like mud, and this was not finally disproved until the 1860s. Controversy surrounding the theory of biological evolution in the late 1800s dissuaded investigation of non-biological origins for life, and at the end of the 1800s it was for example suggested that life on Earth might have arisen from spores of extraterrestrial origin. In the 1920s the idea developed that electrical storms in the atmosphere of the early Earth could lead to production of molecules seen in living systems—and this was confirmed by the experiment of Stanley Miller and Harold Urey in 1953. The molecules obtained were nevertheless still fairly simple—and as it turns out most of them have now also been found in interstellar space. Starting in the 1960s suggestions were made for the chemical and other roles of constituents of the crust as well as atmosphere. Schemes for early forms of self-replication were invented based on molecules such as RNA and on patterns in clay-like materials. (The smallest known system that independently replicates itself is a mycoplasma bacterium with about 580,000 base pairs and perhaps 470 genes. Viroids can be as small as 10,000 atoms but require a host for replication.) In the 1970s it then became popular to investigate complicated cycles of chemical reactions that seemed analogous to ones found in living systems. But with the advent of widespread computer simulations in the 1980s it became clear that all sorts of features normally associated with life were actually rather easy to obtain. (See note above.)

▪ **Page 824 · Self-reproduction.** That one can for example make a mold that will produce copies of a shape has been known since antiquity (see note above). The cybernetics movement highlighted the question of what it takes for self-reproduction to occur autonomously, and in 1952 John von Neumann designed an elaborate 2D cellular automaton that would automatically make a copy of its initial configuration of cells (see page 876). In the course of the 1950s suggestions of several increasingly simple mechanical systems capable of self-reproduction were made—notably by Lionel Penrose. The phenomenon in the main text was noticed around 1961 by Edward Fredkin (see page 877). But while it shows some of the essence of self-reproduction, it lacks many of the more elaborate features common in biological self-reproduction. In the 1980s, however, such features were nevertheless surprisingly often present in computer viruses and worms. (See also page 1092.)

▪ **Page 825 · Extraterrestrial life.** Conditions thermally and chemically similar to those on Earth have presumably existed on other bodies in the solar system. Venus, Mars, Europa (a moon of Jupiter) and Titan (a moon of Saturn) have for example all probably had liquid water at some time. But there is so far no evidence for life now or in the past on any of these. Yet if life had arisen one might expect it to have become widespread, since at least on Earth it has managed to spread to many extremes of temperature, pressure and chemical composition. On several occasions structures have been found in extraterrestrial rocks that look somewhat like small versions of microorganism fossils (most notably in 1996 in a meteorite from Mars discovered in Antarctica). But almost certainly these structures have nothing to do with life, and are instead formed by ordinary precipitation of minerals. And although even up to the 1970s it was thought that life might well be found on Mars, it now seems likely that there is nothing quite like terrestrial life anywhere else in our solar system. (Even if life is found elsewhere it might still have originally come from Earth, say via meteorites, since dormant forms such as spores can apparently survive for long periods in space.)

Away from our solar system there is increasing evidence that most stars have planets with a distribution of sizes—so presumably conditions similar to Earth are fairly common. But thus far it has not been possible to see—say in planetary atmospheres—whether there are for example molecules similar to ones characteristically found in life on Earth.

▪ **Forms of living systems.** This book has shown that even with underlying rules of some fixed type a vast range of different forms can often be produced. And this makes it reasonable to expect that with appropriate genetic programs the chemical building blocks of life on Earth should in principle allow a vast range of forms. But the comparative

weakness of natural selection (see page 391) has meant that only a limited set of such forms have actually been explored. And from the experience of this book I suspect that what others might even be nearby is effectively impossible to foresee. The appearance in engineering of forms somewhat like those in living systems should not be taken to imply that other forms are fundamentally difficult to produce; instead I suspect that it is more a reflection of the copying of living systems for engineering purposes. The overall morphology of living systems on Earth seems to be greatly affected by their basically gelatinous character. So even systems based on solids or gases would likely not be recognized by us as life.

■ **Page 825 · History.** Although Greek philosophers such as Democritus believed that there must be an infinite number of worlds all with inhabitants like us, the prevailing view in antiquity—later supported by theological arguments—was that the Earth is special, and the only abode of life. However, with the development of Copernican ideas in the 1600s it came to be widely though not universally believed, even in theological circles, that other planets—as well as the Moon—must have inhabitants like us. Many astronomers attributed features they saw on the Moon to life if not intelligence, but in the late 1800s, after it was found that the Moon has no atmosphere, belief in life there began to wane. Starting in the 1870s, however, there began to be great interest in life on Mars, and it was thought—perhaps following the emphasis on terrestrial canal-building at the time—that a vast network of canals on Mars had been observed. And although in 1911 the apparent building of new canals on Mars was still being soberly reported by newspapers, there was by the 1920s increasing skepticism. The idea that lichens might exist on Mars and be responsible for seasonal changes in color nevertheless became popular, especially after the discovery of atmospheric carbon dioxide in 1947. Particularly in the 1920s there had been occasional claims of extraterrestrial radio signals (see page 1188), but by the 1950s interest in extraterrestrial intelligence had largely transferred to science fiction (see page 1190). Starting in the late 1940s many sightings were reported of UFOs believed to be alien spacecraft, but by the 1960s these were increasingly discredited. It had been known since the mid-1800s that many other stars are much like the Sun, but it was not until the 1950s that evidence of planets around other stars began to accumulate. Following a certain amount of discussion in the physics community in the 1950s, the first explicit search for extraterrestrial intelligence with a radio telescope was done in 1960 (see page 1189). In the 1960s landings of spacecraft on the Moon confirmed the absence of life there—though returning Apollo astronauts were still quarantined to guard against possible lunar microbes. And despite substantial expectations to the contrary, when spacecraft landed on Mars in 1976 they found no evidence of life there. Some searches for extraterrestrial signals have continued in the radio astronomy community, but perhaps because of its association with science fiction, the topic of extraterrestrial intelligence has generally not been popular with professional scientists. With the rise of amateur science on the web and the availability of low-cost radio telescope components the late 1990s may however have seen a renewal of serious interest.

■ **Page 826 · Bird songs.** Essentially all birds produce calls of some kind, but complex songs are mainly produced by male songbirds, usually in breeding season. Their general form is inherited, but specifics are often learned through imitation during a fixed period of infancy, leading birds in local areas to have distinctive songs. The songs sometimes seem to be associated with attracting mates, and sometimes with defining territory—but often their function is unclear, even when one bird seems to sing in response to another. (There are claims, however, that parrots can learn to have meaningful conversations with humans.) The syrinxes of songbirds have two membranes, which can vibrate independently, in a potentially complex way. A specific region in bird brains appears to coordinate singing; the region contains a few tens of thousands of nerve cells, and is larger in species with more complex songs.

Famous motifs from human music are heard in bird songs probably more often than would be expected by chance. It may be that some common neural mechanism makes the motifs seem pleasing to both birds and humans. Or it could be that humans find them pleasing because they are familiar from bird songs.

■ **Page 826 · Whale songs.** Male whales can produce complex songs lasting tens of minutes during breeding season. The songs often include rhyme-like repeating elements. At a given time all whales in a group typically sing almost the same song, which gradually changes. The function of the song is quite unclear. It has been claimed that its frequencies are optimized for long-range transmission in the ocean, but this appears not to be the case. In dolphins, it is known that one dolphin can produce patterns of sound that are repeated by a specific other dolphin.

■ **Page 826 · Animal communication.** Most animals that live in groups have the capability to produce at least a few specific auditory, visual (e.g. gestures and displays), chemical (e.g. pheromones) or other signals in response to particular situations such as danger. Some animals have also been found to produce much more complex and varied signals.

For example it was discovered in the 1980s that elephants can generate elaborate patterns of sounds—but at frequencies below human hearing. Animals such as octopuses and particularly cuttlefish can show complex and changing patterns of pigmentation. But despite a fair amount of investigation it remains unclear whether these represent more than just simple responses to the environment.

▪ **Page 826 · Theories of communication.** Over the course of time the question of what the essential features of communication are has been discussed from many different angles. It appears to have always been a common view that communication somehow involves transferring thoughts from one mind to another. Even in antiquity it was nevertheless recognized that all sorts of aspects of language are purely matters of convention, so that shared conventions are necessary for verbal communication to be possible. In the 1600s the philosophical idea that the only way to get information with certainty is from the senses led to emphasis on observable aspects of communication, and to the conclusion that there is no way to tell whether an accurate transfer of abstract thoughts has occurred between one mind and another. In the late 1600s Gottfried Leibniz nevertheless suggested that perhaps a universal language—modelled on mathematics—could be created that would represent all truths in an objective way accessible to any mind (compare page 1149). But by the late 1800s philosophers like Charles Peirce had developed the idea that communication must be understood purely in terms of its observable features and effects. Three levels of so-called semiotics were then discussed. The first was syntax: the grammatical or other structure of a sequence of verbal or other elements. The second was semantics: the standardized meaning or meanings of the sequence of elements. And the third was pragmatics: the observable effect on those involved in the communication. In the early 1900s, the logical positivism movement suggested that perhaps a universal language or formalism based on logic could be developed that would allow at least scientific truths to be communicated in an unambiguous way not affected by issues of pragmatics—and that anything that could not be communicated like this was somehow meaningless. But by the 1940s it came to be believed—notably by Ludwig Wittgenstein—that ordinary language, with its pragmatic context, could in the end communicate fundamentally more than any formalized logical system, albeit more ambiguously.

Ever since antiquity work has been done to formalize grammatical and other rules of individual human languages. In the early 1900s—notably with the work of Ferdinand de Saussure—there began to be more emphasis on the general question of how languages really operate, and the point was made that the verbal elements or signs in a language should be viewed as somehow intermediate between tangible entities like sounds and abstract thoughts and concepts. The properties of any given sign were recognized as arbitrary, but what was then thought to be essential about a language is the structure of the network of relations between signs—with the ultimate meaning of any given sign inevitably depending on the meanings of signs related to it (as later emphasized in deconstructionism). By the 1950s anthropological studies of various languages—notably by Benjamin Whorf—had encouraged the idea that concepts that did not appear to fit in certain languages simply could not enter the thinking of users of those languages. Evidence to the contrary (notably about past and future among Hopi speakers) eroded this strong form of the so-called Sapir-Whorf hypothesis, so that by the 1970s it was generally believed just that language can have an influence on thinking—a phenomenon definitely seen with mathematical notation and computer languages. Starting in the 1950s, especially with the work of Noam Chomsky, there were claims of universal features in human languages—independent of historical or cultural context (see page 1103). But at least among linguists these are generally assumed just to reflect common aspects of verbal processing in the human brain, not features that must necessarily appear in any conceivable language. (And it remains unclear, for example, to what extent non-verbal forms of communication such as music, gestures and visual ornament show the same grammatical features as ordinary languages.)

The rise of communications technology in the early 1900s led to work on quantitative theories of communication, and for example in 1928 Ralph Hartley suggested that an objective measure of the information content of a message with n possible forms is $Log[n]$. (Similar ideas arose around the same time in statistics, and in fact there had already been work on probabilistic models of written language by Andrei Markov in the 1910s.) In 1948 Claude Shannon suggested using a measure of information based on $p\,Log[p]$, and there quickly developed the notion that this could be used to find the fundamental redundancy of any sequence of data, independent of its possible meaning (compare page 1071). Human languages were found on this basis to have substantial redundancy (see page 1086), and it has sometimes been suggested that this is important to their operation—allowing errors to be corrected and details of different users to be ignored. (There are also obvious features which reduce redundancy—for example that in most languages common words tend to be short. One can also imagine models of the historical development of languages which will tend to lead to redundancy at the level of Shannon information.)

■ **Mathematical notation.** While it is usually recognized that ordinary human languages depend greatly on history and context, it is sometimes believed that mathematical notation is somehow more universal. But although it so happens that essentially the same mathematical notation is in practice used all around the world by speakers of every ordinary language, I do not believe that it is in any way unique or inevitable, and in fact I think it shows most of the same issues of dependence on history and context as any ordinary language.

As a first example, consider the case of numbers. One can always just use n copies of the same symbol to represent an integer n—and indeed this idea seems historically to have arisen independently quite a few times. But as soon as one tries to set up a more compact notation there inevitably seem to be many possibilities. And so for example the Greek and Roman number systems were quite different from current Hindu-Arabic base-10 positional notation. Particularly from working with computers it is often now assumed that base-2 positional notation is somehow the most natural and fundamental. But as pages 560 and 916 show, there are many other quite different ways to represent numbers, each with different levels of convenience for different purposes. And it is fairly easy to see how a different historical progression might have ended up making another one of these seem the most natural.

The idea of labelling entities in geometrical diagrams by letters existed in Babylonian and Greek times. But perhaps because until after the 1200s numbers were usually also represented by letters, algebraic notation with letters for variables did not arise until the late 1500s. The idea of having notation for operators emerged in the early 1600s, and by the end of the 1600s, notably with the work of Gottfried Leibniz, essentially all the basic notation now used in algebra and calculus had been established. Most of it was ultimately based on shortenings and idealizations of ordinary language, an important early motivation just being to avoid dependence on particular ordinary languages. Notation for mathematical logic began to emerge in the 1880s, notably with the work of Giuseppe Peano, and by the 1930s it was widely used as the basis for notation in pure mathematics.

In its basic structure of operators, operands, and so on, mathematical notation has always been fairly systematic—and is close to being a context-free language. (In many ways it is like a simple idealization of ordinary language, with operators being like verbs, operands nouns, and so on.) And while traditional mathematical notation suffers from some inconsistencies and ambiguities, it was possible in developing *Mathematica* *StandardForm* to set up something very close that can be interpreted uniquely in all cases.

Mathematical notation works well for things like ordinary formulas that involve a comparatively small number of basic operations. But there has been no direct generalization for more general constructs and computations. And indeed my goal in designing *Mathematica* was precisely to provide a uniform notation for these (see page 852). Yet to make this work I had to use names derived from ordinary language to specify the primitives I defined.

■ **Computer communication.** Most protocols for exchanging data between computers have in the end traditionally had rather simple structures—with different pieces of information usually being placed at fixed positions, or at least being arranged in predefined sequences—or sometimes being given in name-value pairs. A more general approach, however, is to use tree-structured symbolic expressions of the kind that form the basis for *Mathematica*—and now in essence appear in XML. In the most general case one can imagine directly exchanging a representation of a program, that is run on the computer that receives it, and induces whatever effect one wants. A simple example from 1984 is *PostScript*, which can specify a picture by giving a program for constructing it; a more sophisticated example from the late 1990s is client-side Java. (Advanced forms of data compression can also be thought of as working by sending simple programs.) But a practical problem in exchanging arbitrary programs is the difficulty of guarding against malicious elements like viruses. And although at some level most communications between present-day computers are very regular and structured, this is often obscured by compression or encryption.

When a program is sent between computers it is usually encoded in a syntactically very straightforward way. But computer languages intended for direct use by humans almost always have more elaborate syntax that is a simple idealization of ordinary human language (see page 1103). There are in practice many similarities between different computer languages. Yet except in very low-level languages few of these are necessary consequences of features or limitations of actual computers. And instead most of them must be viewed just as the results of shared history—and the need to fit in with human cognitive abilities.

■ **Meaning in programs.** Many issues about meaning arise for computer languages in more defined versions of the ways they arise for ordinary languages. Input to a computer language will immediately fail to be meaningful if it does not conform to a certain definite syntax. Before the input can have a chance of specifying meaningful action there are often all sorts of issues about whether variables in it refer to entities that can be considered to exist. And even if this is resolved, one can still get something that is in effect nonsense and does

not usefully run. In most traditional computer languages it is usually the case that most programs chosen at random will just crash if run, often as a result of trying to write to memory outside the arrays they have allocated. In *Mathematica*, there is almost no similar issue, and programs chosen at random tend instead just to return unchanged. (Compare page 101.)

For the kinds of systems like cellular automata that I have discussed in this book programs chosen at random do very often produce some sort of non-trivial behavior. But as discussed in the main text there is still an issue of when this behavior can reasonably be considered meaningful.

For some purposes a more direct analog of messages is not programs or rules for systems like cellular automata but instead initial conditions. And one might imagine that the very process of running such initial conditions in a system with appropriate underlying rules would somehow be what corresponds to their meaning. But if one was just given a collection of initial conditions without any underlying rules one would then need to find out what underlying rules one was supposed to use in order to determine their meaning. Yet the system will always do something, whatever rules one uses. So then one is back to defining criteria for what counts as meaningful behavior in order to determine—by a kind of generalization of cryptanalysis—what rules one is supposed to use.

▪ **Meaning and regularity.** If one considers something to show regularity one may or may not consider it meaningful. But if one considers something random then usually one will also consider it meaningless. For to say that something is meaningful normally implies that one somehow comes to a conclusion from it. And this typically implies that one can find some summary of some aspect of it—and thus some regularity. Yet there are still cases where things that are presumably quite random are considered meaningful—prices in financial markets being one example.

▪ **Page 828 · Forms of artifacts.** Much as in biological evolution, once a particular engineering construct has been found to work it normally continues to be used. Examples with characteristic forms include (in rough order of their earliest known use): arrowheads, boomerangs, saws, boxes, stairs, fishhooks, wheels, arches, forks, balls, kites, lenses, springs, catenaries, cogs, screws, chains, trusses, cams, linkages, propellers, clocksprings, parabolic reflectors, airfoils, corrugation, zippers, and geodesic domes. It is notable that not even nested shapes are common, though they appear in cross-sections of rope (see page 874), as well as in address decoder trees on chips—and have recently been used in broadband antennas. (Some self-similarity is also present in standard log-periodic antennas.) When several distinct components are involved, more complicated structures are not uncommon—as in escapements, and many bearings and joints. More complex shapes for single elements sometimes arise when an analog of area maximization is desired—as with tire treads or fins in devices such as heat exchangers. Quadratic residue sequences `Mod[Range[n]^2, n]` (see page 1081) are used to give profiles for acoustic diffusers that operate uniformly over a range of frequencies. Musical instruments can have fairly complicated shapes maintained for historical reasons to considerable precision. Some knots can also be thought of as objects with complex forms. It is notable that elaborate types of mechanical motion (and sometimes surprising phenomena in general) are often first implemented in toys. Examples are early mechanical automata and model airplanes, and modern executive toys claiming to illustrate chaos theory through linkages, magnets or fluid systems. Complex trajectories (compare page 972) have sometimes been proposed or used for spacecraft. (See also notes on ornamental art on page 872.)

▪ **Page 828 · Recognizing artifacts.** Various situations require picking out artifacts automatically. One example is finding buildings or machines from aerial reconnaissance images; another is finding boat or airplane wreckage on an ocean floor from sonar data. In both these cases the most common approach is to look for straight edges. Outdoor security systems also often need ways to distinguish animals and wind-induced motion from intentional human activity—and tend to have fairly simple procedures for doing this.

To recognize a regular crystal as not being a carefully cut artifact can take specific knowledge. The same can be true of patterns produced by wind on sand or rocks. Lenticular clouds are sometimes mistaken for UFOs on account of their regular shape. The exact cuboid form of the monolith in the movie *2001* was intended to suggest that it was an artifact.

Recognizing artifacts can be a central—and controversial—issue in prehistoric archeology. Sometimes human bones are found nearby. And sometimes chemical analysis suggests controlled fire—as with charcoal or baked clay. But to tell whether for example a piece of rock was formed naturally or was carefully made to be a stone tool can in general be very difficult. And a large part of the way this has been done in practice is just through comparison with known examples that fit into an overall pattern of gradual historical change. In recent decades there has been increasing emphasis on trying to understand and reproduce the whole process of making and using artifacts. And in the field of lithic analysis there are beginning to be fairly systematic ways to recognize for example the effects of the hundreds of orderly impacts needed to make a typical flint arrowhead by knapping. (Sometimes it is also possible to recognize microscopic features characteristic

of particular kinds of use or wear—and it is conceivable that in the future analysis of trillions of atomic-scale features could reveal all sorts of details of the history of an object.)

To tell whether or not some arrangement of soil or rocks is an artifact can be extremely difficult—and there are many notorious cases of continuing controversy. Beyond looking for similarities to known examples, a typical approach is just to look for correlations with topographic or other features that might reveal some possible purpose.

■ **Artifacts in data.** In fields like accounting and experimental science it is usually a sign of fraud if primary data is being created for a purpose, rather than merely being reported. If a large amount of numerical data has been made up by a person this can be detectable through statistical deviations from expected randomness—particularly in structural details such as frequency of digits. (So-called artifacts can also be the unintentional result of details of methods used to obtain or process data.)

In numerical computations effects are often called artifacts if they are believed not to be genuine features of an underlying mathematical system, but merely to reflect the computational scheme used. Such effects are usually first noticed through unexpected regularities in some detail of output. But in cases like chaos theory it remains unclear to what extent complex behavior seen in computations is an artifact (see page 920).

■ **Animal artifacts.** Structures like mollusc shells, radiolarian skeletons and to some extent coral are formed through processes of growth like those discussed in Chapter 8. Structures like spider webs, wasp nests, termite mounds, bird nests and beaver dams rely on behavior determined by animal brains. (Even spider webs end up looking quite different if psychoactive drugs are administered to the spider.) And much like human artifacts, many of these structures tend to be distinguished by their comparative geometrical simplicity. In a few cases—particularly with insects—somewhat complicated forms are seen, but it seems likely that these are actually produced by rather simple local rules like those in aggregation systems (see page 1011).

■ **Molecular biology.** DNA sequences of organisms can be thought of as artifacts created by biological evolution, and current data suggests that they contain some long-range correlations not present in typical random sequences. Most likely, however, these have fairly simple origins, perhaps being associated with iterative splicing of subsequences. And in the few thousand proteins currently known, standard statistical tests reveal no significant overall regularities in their sequences of up to a few thousand amino acids. (Some of the 20 standard amino acids do however occur more frequently than others.) Nevertheless, if one looks at overall shapes into which these proteins fold, there is some evidence that the same patterns of behavior are often seen. But probably such patterns would also occur in purely random proteins—at least if their folding happened in the same cellular apparatus. (See page 1003.) Note that the antibodies of the immune system are much like short random proteins—whose range of shapes must be sufficient to match any antigen. (See also page 1194.)

■ **Messages in DNA.** Science fiction has sometimes suggested that an extraterrestrial source of life might have left some form of message in the DNA sequences of all terrestrial organisms, but to get evidence of this would seem to require extensive other knowledge of the source. (See also page 1190.)

■ **Decompilers.** Trying to reverse engineer source code in a programming language like C from machine code output by compilers involves in effect trying to deduce higher-level purposes from lower-level computational steps. And normally this can be done with any reliability only when the machine code represents a fairly direct translation that has not been extensively rearranged or optimized.

■ **Page 828 · Complexity and theology.** See page 861.

■ **Page 829 · Purpose in archeology.** Ideas about the purpose of archeological objects most often ultimately tend to come from comparisons to similar-looking objects in use today. But great differences in typical beliefs and ways of life can make comparisons difficult. And certainly it is now very hard for us to imagine just what range of purposes the first known stone tools from 2.6 million years ago might have been put to—or what purpose the arrays of dots or handprints in cave paintings from 30,000 BC might have had. And even when it comes to early buildings from perhaps 10,000 BC it is still difficult to know just how they were used. Stone circles like Stonehenge from perhaps 3000 BC presumably served some community purpose, but beyond that little can convincingly be said. Definite geographical or astronomical alignments can be identified for many large prehistoric structures, but whether these were actually intentional is almost never clear. After the development of writing starting around 4000 BC, purposes can often be deduced from inscriptions and other written material. But still to work out for example the purpose of the Antikythera device from around 100 BC is very difficult, and depends on being able to trace a long historical tradition of astronomical clocks and orreries.

■ **Dead languages.** Particularly over the past century or so, most of the known written human languages from every point in history have successfully been decoded. But to do this has essentially always required finding a case where there is explicit overlap with a known language—say a

Rosetta stone with the same text in multiple languages, or at least words or proper names that are transliterated. As in cryptanalysis, it is sometimes remarkable how small an amount of text is needed to find a decoding scheme. But usually what is done relies critically on the slowness with which human languages change, and the comparatively limited number of different basic ways in which they work.

■ **Teleology.** There is a common tendency to project human purposes onto natural objects and events—and this is for example almost universally done by young children. Ancient beliefs often held that things in nature are set up by a variety of gods for a variety of purposes. By 400 BC, following ideas of Anaxagoras, Socrates and Plato discussed the notion that things in nature might in effect be optimally designed for coherent purposes by a single mind. Around 350 BC Aristotle claimed that a full explanation of anything should include its purpose (or so-called final cause, or telos)—but said that for systems in nature this is often just to make the final forms of these systems (their so-called formal cause). The rise of monotheistic religions led to the widespread belief that the universe and everything in it was created for definite purposes by a single god. But the development of mathematical science in the 1600s—and its focus on mechanisms ("efficient causes")—led away from ascribing explicit purposes to physical systems. In the mid-1700s David Hume then claimed on philosophical grounds that we fundamentally have no basis for ascribing purposes to any kind of natural system—though in the 1790s Immanuel Kant argued that even though we cannot know whether there really are such purposes, it is still often necessary for us to think in terms of them. And in fact the notion that systems in biology are so complex that they must have been intelligently designed for a purpose remained common. In the late 1800s Darwinian evolution nevertheless suggested that no such purposeful design was necessary—though in a sense it again introduced a notion of purpose associated with optimization of fitness. Ever since the 1700s economics had been discussed in terms of purposeful activities of rational agents. In the early 1900s there were however general attempts to develop mechanistic explanations in the social sciences, but by the mid-1900s purpose was again widely discussed, especially in economics. And in fact, even in physics, a notion of purpose had actually always been quite common. For whenever a physical system satisfies any kind of implicit equation, this defines a constraint that can be viewed as corresponding to some kind of purpose. (See page 940.) That something like a notion of purpose is being used has been more widely recognized for variational principles like the Principle of Least Action in mechanics from the mid-1700s. Results in the late 1900s in astrophysics and cosmology seemed to suggest that for us to exist our universe must satisfy all sorts of constraints—and to avoid explaining this in terms of purpose the Anthropic Principle was introduced (see page 1026). What I do in this book goes significantly further than traditional science in getting rid of notions of purpose from investigations of nature. For I essentially always consider systems that are based on explicit evolution rules rather than implicit constraints. And in fact I argue that simple programs constructed without known purposes are what one needs to study to find the kinds of complex behavior we see.

■ **Possible purposes.** As part of asking whether the rules for a system are somehow minimal for a given purpose, one can ask what properties the system has that could reasonably be considered a purpose at all. In general one tends to talk of purpose only when doing so allows one to give a simpler description of some aspect of behavior than just describing the behavior directly. But whether one can give a simple description can depend greatly on the framework in which one is operating. And so, for example, while the digits of π have a simple description in terms of traditional mathematics, the results in Chapter 4 suggest that outside of this framework they normally do not. And what this means is that if one saw a system that had the property of generating the digits of π one would be unlikely to think that this could represent a meaningful purpose—unless one happened to be operating in traditional mathematics. And so similarly, one would be unlikely to think that generating the center column from rule 30 could represent any sort of meaningful purpose—unless one was operating within the framework that I have developed in this book.

■ **Page 830 · Purposeful computation.** See page 638.

■ **Page 832 · Doubling rules.** Rule (a) is

```
{{0, 2, _} → 5, {5, 3, _} → 5, {5, _, _} → 1,
  {_, 5, _} → 1, {_, 2, _} → 3, {_, 3, 2} → 2, {_, 1, 2} → 4,
  {_, 4, _} → 3, {4, 3, _} → 4, {4, 0, _} → 2, {_, x_, _} → x}
```

and takes $2n^2 + n$ steps to yield *Table[1, {2 n}]* given input *Append[Table[1, {n - 1}], 2]*. Rule (b) is

```
{{_, 2, _} → 3, {_, 1, 2} → 2, {3, 0, _} → 1,
  {3, _, _} → 3, {_, 3, _} → 1, {_, x_, _} → x}
```

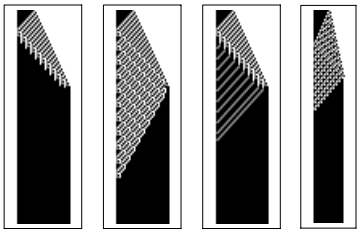
and takes $3n$ steps. Rule (c) is $k = 3$, $r = 1$ rule *5407067979* and takes $3n - 1$ steps.

■ **Page 833 · Searching.** No symmetric $k = 3$, $r = 1$ rule yields doubling. General rules can show subtle bugs; rule *1340716537107* for example first fails at $n = 24$. The total number of $k = 3$, $r = 1$ rules that need to be searched can easily be reduced from 3^{27} to 3^{21}. Several different rules that work can behave identically, since up to 6 of the 27 cases in each rule are not sampled with the initial conditions used. In

rules that work, between 8 and 19 cases lead to a change in the color of a cell, with 14 cases being the most common.

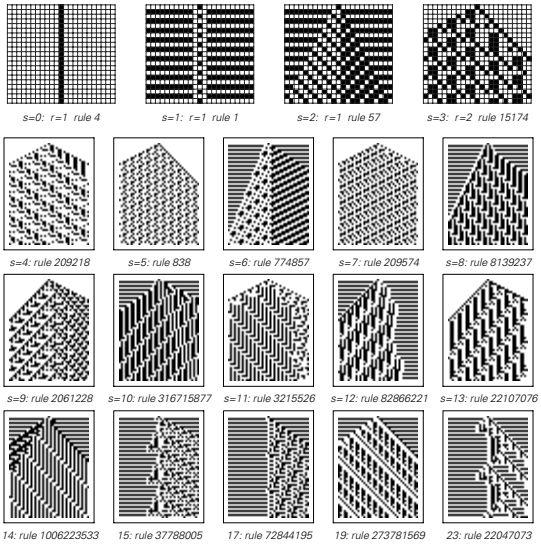
■ **Page 833 · Properties.** The number of steps increases irregularly but roughly quadratically with n in rule (a), and roughly linearly in (d) and (e). Rule (b) in the end repeats every 128 steps. The center of the complex pattern in both (d) and (e) emulates $k = 2$ rule 90.

■ **Other functions.** The first three pictures below show rules that yield $3n$ (no $k = 3$ rules yield $4n$, $5n$ or n^2), and the last picture $2n-2$ (corresponding to doubling with initial conditions analogous to page 639).



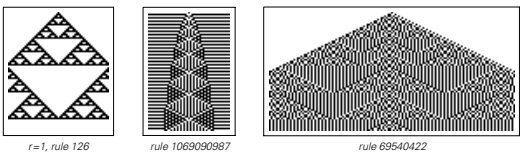
304911688608 308527554123 1183925790477 2672802162657

■ **Page 834 · Minimal cellular automata for sequences.** Given any particular sequence of black and white cells one can look for the simplest cellular automaton which generates that sequence as its center column when evolving from a single black cell (compare page 956). The pictures below show the lowest-numbered cellular automaton rules that manage to generate repetitive sequences containing black cells with successively greater separations s.



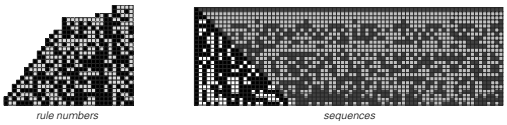
Elementary ($k = 2$, $r = 1$) cellular automata can be found only up to separations $s = 2$. But $k = 2$, $r = 2$ cellular automata can be found for all separations up to 15, as well as 17, 19 and 23. (Note that for example in the $s = 15$ case the lowest-numbered rule exhibits a complex 350-step transient away from the center column.)

The pictures below show the lowest-numbered cellular automata that generate respectively powers of two, squares and the nested Thue-Morse sequence of page 83 (compare rule 150). Of the 4 billion $k = 2$, $r = 2$ cellular automata none turn out to be able to produce for example sequences corresponding to the cubes, powers of 3, Fibonacci numbers, primes, digits of $\sqrt{2}$, or concatenation sequences.



r=1, rule 126 rule 1069090987 rule 69540422

If one looks not just at specific sequences, but instead at all 2^n possible sequences of length n, one can ask how many cellular automaton rules (say with $k = 2$, $r = 2$) one has to go through in order to generate every one of these. The pictures below show on the left the last rules needed to generate any sequence of each successive length—and on the right the form of the sequence (as well as its continuation after length n). Since some different rules generate the same sequences (see page 956) one needs to go through somewhat more than 2^n rules to get every sequence of length n. The sequences shown below can be thought of as being in a sense the ones of each length that are the most difficult to generate—or have the highest algorithmic information content. (Note that the sequence ■■□ is the first one that cannot be generated by any of the 256 elementary cellular automata; the first sequence that cannot be generated by any $k = 2$, $r = 2$ cellular automata is probably of length 26.)



rule numbers sequences

■ **Other examples.** Minimal systems achieving particular purposes are shown on page 619 for Boolean functions evaluated with NANDs, pages 759 and 889 for Turing machines, page 1142 for sorting networks, and page 1035 for firing squad synchronization.

■ **Page 834 · Minimal theories.** Particularly in fundamental physics it has been found that the correct theory is often the minimal one consistent with basic observations. Yet barring

supernatural intervention, the laws of physics embodied in such a theory presumably cannot be considered to have been created for any particular purpose. (See page 1025.)

▪ **Page 835 · Earth from space.** Human activity has led to a few large simple geometrical structures that are visually noticeable from space. One is the almost-straight 30-mile railroad causeway built in 1959 that divides halves of the Great Salt Lake in Utah where the water is colored blue and orange. Another is the almost-circular 12-mile-diameter national park created in 1900 that encloses ungrazed vegetation on the Egmont Volcano in New Zealand. On the scale of a few miles, there is also rectilinear arrangement of fields in the U.S. Midwest, as well as straight-line political boundaries with different agriculture on each side. Large geometrical patterns of logging were for example briefly visible after snow in 1961 near Cochrane, Canada—as captured by an early weather satellite. Perfectly straight sections of roads (such as the 90-mile Balladonia-Caiguna road in Australia), as well as the 4-mile-diameter perfectly circular Fermilab accelerator ring are not so easy to see. The Great Wall of China from 200 BC follows local topography and so is not straight.

Some of the most dramatic geometrical structures—such as the dendritic fossil drainage pattern in south Yemen or the bilaterally symmetric coral reefs around islands like Bora Bora—are not artifacts. The same is true of fields of parallel sand dunes, as well as of almost-circular structures such as the 40-mile-diameter impact crater in Manicouagan, Canada (highlighted by an annular lake) and the 30-mile-diameter Richat structure in the Sahara desert of Mauritania. On the Moon, the 50-mile-diameter crater Tycho is also almost circular—and has 1000-mile almost-straight rays coming out from it.

At night, lights of cities are obvious—notably hugging the coast of the Mediterranean—as are fire plumes from oil rigs. In addition, in some areas, sodium streetlamps make the light almost monochromatic. But it would seem difficult to be sure that these were artifacts without more information. In western Kansas there is however a 200-mile square region with light produced by a strikingly regular grid of towns—many at the centers of square counties laid out around 1870 in connection with land grants for railroad development. In addition, there is an isolated 1000-mile straight railroad built in the late 1800s across Kazakhstan between Aktyubinsk and Tashkent, with many towns visible at night along it. There are also 500-mile straight railroads built around the same time between Makat and Nukus, and Yaroslavl and Archangel. All these railroads go through flat empty terrain that previously had only a few nomadic inhabitants—and no settlements to define a route. But in many ways such geometrical forms seem vastly simpler to imagine producing than for example the elaborate pattern of successive lightning strikes visible especially in the tropics from space.

▪ **Page 835 · Astronomical objects.** Stars and planets tend to be close to perfect spheres. Lagrange points and resonances often lead to simple geometrical patterns of orbiting bodies. (The orbits of most planets in our solar system are also close to perfect circles; see page 973.) Regular spirograph-like patterns can occur for example in planetary nebulas formed by solar mass exploding stars. Unexplained phenomena that could conceivably be at least in part artifacts include gamma ray bursts and ultra high-energy cosmic rays. The local positions of stars are generally assumed to be random. 88 constellations are usually named—quite a few presumably already identified by the Babylonians and Sumerians around 2000 BC.

▪ **Page 835 · Natural radio emissions.** Each of the few million lightning flashes that occur on the Earth each day produce bursts of radio energy. At kilohertz frequencies reflection from the ionosphere allows these signals to propagate up to thousands of miles around the Earth, leading to continual intermittent crackling and popping. Particularly at night such signals can also travel within the ionosphere, but different frequencies travel at different rates, leading to so-called tweeks involving ringing or pinging. Signals can sometimes travel through the magnetosphere along magnetic field lines from one hemisphere to the other, yielding so-called whistlers with frequencies that fall off in a highly regular way with time. (Occasionally the signals can also travel back and forth between hemispheres, giving more complex results.) Radio emission can also occur when charged particles from the Sun excite plasma waves in the magnetosphere. And particularly at dawn or when an aurora is present an elaborate chorus of different elements can be produced—and heard directly on a VLF radio receiver.

Sunspots and solar flares make the Sun the most intense radio source in the solar system. Artificial radio signals from the Earth come next. The interaction of the solar wind with the magnetosphere of Jupiter produces radio emissions that exhibit variations reminiscent of gusting.

Outside the solar system, gas clouds show radio emission at discrete gigahertz frequencies from rotational transitions in molecules and spin-flip transitions in hydrogen atoms. (The narrowest lines come from natural masers and have widths around 1 kHz.) The cosmic microwave background, and processes such as thermal emission from dust, radiation from electrons in ionized gases, and synchrotron radiation from relativistic electrons in magnetic fields yield radio emissions

with characteristic continuous frequency spectra. A total of over a million radio sources inside and outside our galaxy have now been catalogued, most with frequency spectra apparently consistent with known natural phenomena. Variations of source properties on timescales of months or years are not uncommon; variations of signals on timescales of tens of minutes can be introduced by propagation through turbulence in the interstellar medium.

Most radio emission from outside the solar system shows little apparent regularity. The almost perfectly repetitive signals from pulsars are an exception. Pulsars appear to be rapidly rotating neutron stars—perhaps 10 miles across—whose magnetic fields trap charged particles that produce radio emissions. When they first form after a supernova pulsars have millisecond repetition rates, but over the course of a few million years they slow to repetition rates of seconds through a series of glitches, associated perhaps with cracking in their solid crusts or perhaps with motion of quantized vortices in their superfluid interiors. Individual pulses from pulsars show some variability, presumably largely reflecting details of plasma dynamics in their magnetospheres.

■ **Page 835 · Artificial radio signals.** In current technology radio signals are essentially always based on carriers of the form $Sin[\omega t]$ with frequencies $\omega/(2\pi)$. When radio was first developed around 1900 information was normally encoded using amplitude modulation (AM) $s[t] Sin[\omega t]$. In the 1940s it also became popular to use frequency modulation (FM) $Sin[(1 + s[t]) \omega t]$, and in the 1970s pulse code modulation (PCM) (pulse trains for *IntegerDigits[s[t], 2]*). All such methods yield signals that remain roughly in the range of frequencies $\{\omega - \delta, \omega + \delta\}$ where δ is the data rate in $s[t]$. But in the late 1990s—particularly for the new generation of cellular telephones—it began to be common to use spread spectrum CDMA methods, in which many signals with the same carrier frequency are combined. Each is roughly of the form *BitXor[u[t], s[t]] Sin[ω t]*, where *u[t]* is a pseudonoise (PN) sequence generated by a linear feedback shift register (LFSR) (see page 1084); the idea is that by using a different PN sequence for each signal the corresponding *s[t]* can be recovered even if thousands are superimposed.

The radio spectrum from about 9 kHz to 300 GHz is divided by national and international legislation into about 460 bands designated for different purposes. And except when spread spectrum methods are used, most bands are then divided into between a few and a few thousand channels in which signals with identical structures but different frequencies are sent.

If one steps through frequencies with an AM radio scanner, one sometimes hears intelligible speech—from radio or TV broadcasts, or two-way radio communication. But in many frequency bands one hears instead either very regular or seemingly quite random signals. (A few bands allocated for example to distress signals or radio astronomy are normally quiet.) The regular signals come from such sources as navigation beacons, time standards, identification transponders and radars. Most have characteristic almost perfectly repetitive forms (radar pulses, for example, typically have the chirped form $Sin[(1 + \alpha t) \omega t]$)—and some sound uncannily like pulsars. When there are seemingly random signals some arise say from transmission of analog video (though this typically has very rigid overall structure associated with successive lines and frames), but most are now associated with digital data. And when CDMA methods are used there can be spreading over a significant range of frequencies—with regularities being recognizable only if one knows or can cryptanalyze LFSR sequences.

In general to send many signals together one just needs to associate each with a function *f[i, t]* orthogonal to all other functions *f[j, t]* (see page 1072). Current electronics (with analog elements such as phase-locked loops) make it easy to handle functions $Sin[\omega t]$, but other functions can yield better data density and perhaps better signal propagation. And as faster digital electronics makes it easier to implement these it seems likely that it will become less and less common to have simple carriers with definite frequencies.

In addition, there is a continuing trend towards greater spatial localization of signals—whether by using phased arrays or by explicitly using technologies like fiber optics.

At present, the most intense overall artificial radio emission from the Earth is probably the 50 or 60 Hz hum from power lines. The most intense directed signals are probably from radars (such as those used for ballistic missile detection) that operate at a few hundred megahertz and put megawatts of power into narrow beams. (Some such systems are however being replaced by lower-power phased array systems.)

■ **Page 835 · SETI.** First claims of extraterrestrial radio signals were made by Nikola Tesla in 1899. More widely believed claims were made by Guglielmo Marconi in 1922, and for several years searches were done—notably by the U.S. military—for signals presumed to be coming from Mars. But it became increasingly accepted that in fact nothing beyond natural radio emissions such as whistlers (see note above) were actually being detected.

When galactic radio emission was first noticed by Karl Jansky in 1931 it seemed too random to be of intelligent origin. And when radio astronomy began to develop it essentially ignored extraterrestrial intelligence. But in 1959

Giuseppe Cocconi and Philip Morrison analyzed the possibility of interstellar radio communication, and in 1960 Frank Drake used a radio telescope to look for explicit signals from two nearby stars.

In 1965 a claim was made that there might be intensity variations of intelligent origin in radio emission from the quasar CTA-102—but this was quickly retracted. Then in 1967 when the first pulsar was discovered it was briefly thought that perhaps its precise 1.33730113-second repetition rate might be of intelligent origin.

Since the 1960s around a hundred different SETI (search for extraterrestrial intelligence) experiments have been done. Most use the same basic scheme: to look for signals that show a narrow band of frequencies—say only 1 Hz wide—perhaps changing in time. (The corresponding waveform is thus required to be an almost perfect sinusoid.) Some concentrate on specific nearby stars, while others look at the whole sky, or test the stream of data from all observations at a particular radio telescope, sometimes scanning for repetitive trains of pulses rather than single frequencies. The best current experiments could successfully detect radio emission at the level now produced on Earth only from about 10 light years away—or from about the nearest 10 stars. The detection distance increases like the square root of the signal strength, covering all 10^{11} stars in our galaxy when the signal uses the total power output of a star.

Most SETI has been done with specially built systems or with existing radio telescopes. But starting in the mid-1990s it became possible to use standard satellite receivers, and there are now plans to set up a large array of these specifically for SETI. In addition, it is now possible to use software instead of hardware to implement SETI signal-processing algorithms—both traditional ones and presumably much more general ones that can for example pick out much weaker signals.

Many SETI experiments look for signals in the so-called "water hole" between the 1420 MHz frequency associated with the 21 cm line of hydrogen and the 1720 MHz frequency associated with hydroxyl (OH). But although there are now practical constraints associated with the fact that on Earth only a few frequency regions have been left clear for radio astronomy I consider this to be a remarkable example of reliance on details of human intellectual development.

Already in the early 1960s it was suggested that lasers instead of radio could be used for interstellar communication, and there have been various attempts to detect interstellar optical pulses. Other suggested methods of communication have included optical solitons, neutrinos and as-yet-unknown faster-than-light quantum effects.

It is sometimes suggested that there must be fundamental limits to detection of radio signals based on such issues as collection areas, noise temperatures and signal degradation. But even existing technology has provided a steady stream of examples where limits like these have been overcome—most often by the use of more sophisticated signal processing.

▪ **Detection methods.** Ways to identify computational origins include looking for repeatability in apparently random signals and comparing with output from large collections of possible simple programs. At a practical level, the one-dimensional character of data from radio signals makes it difficult for us to apply our visual systems—which remain our most powerful general-purpose analysis tools.

▪ **Higher perception and analysis.** See page 632.

▪ **Page 837 · Messages to send.** The idea of trying to send messages to extraterrestrials has existed since at least the early 1800s. The proposed content and medium of the messages has however steadily changed, usually reflecting what seemed to be the most significant human achievements of the time—yet often seeming quaint within just a few decades.

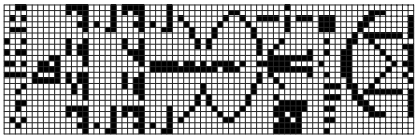
Starting in the 1820s various scientists (notably Carl Friedrich Gauss) suggested signalling the Moon by using such schemes as cutting clearings in a forest to illustrate the Pythagorean theorem or reflecting sunlight from mirrors in different countries placed so as to mimic an observed constellation of stars. In the 1860s, with the rise of telegraphy, schemes for sending flashes of light to Mars were discussed, and the idea developed that mathematics should somehow be the basic language used. In the 1890s radio signals were considered, and were tried by Nikola Tesla in 1899. Discussion in the 1920s led to the idea of sending radio pulses that could be assembled into a bitmap image, and some messages intended for extraterrestrials were probably sent by radio enthusiasts.

There is a long history of attempts to formulate universal languages (see page 1181). The Lincos language of Hans Freudenthal from 1960 was specifically designed for extraterrestrial communication. It was based on predicate logic, and attempted to use this to build up first mathematics, then science, then a general presentation of human affairs.

When the Pioneer 10 spacecraft was launched in 1972 it carried a physical plaque designed by Carl Sagan and others. The plaque is surprisingly full of implicit assumptions based on details of human intellectual development. For example, it has line drawings of humans—whose interpretation inevitably seems very specific to our visual system and artistic culture. It also has a polar plot of the positions of 14 pulsars relative to the Sun, with the pulsars specified by giving their periods as base 2 integers—but with trailing

zeros inserted to cover inadequate precision. Perhaps the most peculiar element, however, is a diagram indicating the 21 cm transition in hydrogen—by showing two abstract quantum mechanical spin configurations represented in a way that seems completely specific to the particular details of human mathematics and physics. In 1977 the Voyager spacecraft carried phonograph records that included bitmap images and samples of spoken languages and music.

In 1974 the bitmap image below was sent as a radio signal from the Arecibo radio telescope. At the left-hand end is a version of the pattern of digits from page 117—but distorted so it has no obvious nested structure. There follow atomic numbers for various elements, and bitvectors for components of DNA. Next are idealized pictures of a DNA molecule, a human, and the telescope. All these parts seem to depend almost completely on detailed common conventions—and I suspect that without all sorts of human context their meaning would be essentially impossible to recognize.



In all, remarkably few messages have been sent—perhaps in part because of concerns that they might reveal us to extraterrestrial predators (see page 1191). There has also been a strong tendency to make messages hard even for humans to understand—perhaps on the belief that they must then be more scientific and more universal.

The main text argues that it will be essentially impossible to give definitive evidence of intelligence. Schemes that might however get at least some distance include sending:

- waveforms made of simple underlying elements;
- long complicated sequences that repeat precisely;
- a diversity of kinds of sequences;
- something complicated that satisfies simple constraints.

Examples of the latter include pattern-avoiding sequences (see page 944), magic squares and other combinatorial designs, specifications of large finite groups, and maximal length linear feedback shift register sequences (see page 1084). Notably, the last of these are already being transmitted by GPS satellites and CDMA communications systems. (If cases could be found where the sequences as a whole were forced not to have any obvious regularities, then pattern-avoiding sequences might perhaps be good since they have constraints that are locally fairly easy to recognize.)

Extrapolation of trends in human technology suggest that it will become ever easier to detect weak signals that might be assumed distorted beyond recognition or swamped by noise.

▪ **Page 838 · P versus NP.** Given a constraint, it may be an NP-complete problem to find out what object satisfies it. So it may be difficult to generate the object from the constraint. But if one allows oneself to generate the object in any way at all, this may still be easy, even if $P \neq NP$.

▪ **Science fiction.** Inhabitants of the Moon were described in stories by Lucian around 150 AD and Johannes Kepler in 1634—and in both cases were closely modelled on terrestrial organisms. Interest in fiction about extraterrestrials increased greatly at the end of the 1800s—perhaps because by then few parts of the Earth remained unexplored. And as science fiction developed, accounts of the future sometimes treated extraterrestrials as commonplace—and sometimes did not mention them at all. Most often extraterrestrials have been easy to recognize, being little more than simple combinations of terrestrial animals (and occasionally plants)—though fairly often with extra features like telepathy. Some stories have nevertheless explored extraterrestrial intelligence based for example on solids, gases or energy fields. An example is Fred Hoyle's 1957 *The Black Cloud* in which a large cloud of hydrogen gas achieves intelligence by exchanging electromagnetic signals between rocks whose surface molecular configurations store memories.

The most common fictional scenario for first contact with extraterrestrials is the arrival of spacecraft—often induced by us having passed a technology threshold such as radio, nuclear explosions or faster-than-light travel. Other scenarios sometimes considered include archeological discovery of extraterrestrial artifacts and receipt of radio signals.

In the movie *2001* a black cuboid with side ratios 1:4:9 detected on the Moon through its anomalous magnetic properties sends a radio pulse in response to sunlight. Later there are also a few frames of flashing octahedra, presumably intended to be extraterrestrial artifacts, or perhaps extraterrestrials themselves.

In *The Black Cloud* intelligence is suggested by responsiveness to radio stimuli. Communication is established—as often in science fiction—by the intelligence interpreting material that we supply, and then replying in the same format.

The movie *Contact* centers on a radio signal with several traditional SETI ideas: it is transmitted at $1420\,\pi$ MHz, and involves a sequence of primes to draw attention, an amplified TV signal from Earth and a description of a machine to build.

The various *Star Trek* television series depict many encounters with "new life and new civilizations". Sometimes intelligence is seen not associated with something that is considered a lifeform.

Particularly in short stories various scenarios have been explored where it is difficult ever to recognize intelligence. These include one-of-a-kind beings that have nothing to communicate with, as well as beings with inner intellectual activity but no effect on the outside world. When there are extraterrestrials substantially more advanced than humans few efforts have been made to describe their motives and purposes directly—and usually what is emphasized is just their effects on humans.

(See also page 1184.)

▪ **Page 839 · Practical arguments.** If extraterrestrials exist at all an obvious question—notably asked by Enrico Fermi in the 1940s—is why we have not encountered them. For there seems no fundamental reason that even spacecraft could not colonize our entire galaxy within just a few million years.

Explanations suggested for apparent absence include:

- Extraterrestrials are visiting, but we do not detect them;
- Extraterrestrials have visited, but not in recorded history;
- Extraterrestrials choose to exist in other dimensions;
- Interstellar travel is somehow infeasible;
- Colonization is somehow ecologically limited;
- Physical travel is not worth it; only signals are ever sent.

Explanations for apparent lack of radio signals include:

- Broadcasting is avoided for fear of conquest;
- There are active efforts to prevent us being contaminated;
- Extraterrestrials have no interest in communicating;
- Radio is the wrong medium;
- There are signals, but we do not understand them.

The so-called Drake equation gives a straightforward upper bound on the number of cases of extraterrestrial intelligence that could have arisen in our galaxy through the same basic chain of circumstances as humans. The result is a product of: rate of formation of suitable stars; fraction with planetary systems; number of Earth-like planets per system; fraction where life develops; fraction where intelligence develops; fraction where technology develops; time communicating civilizations survive. It now seems fairly certain that there are at least hundreds of millions of Earth-like planets in our galaxy. Biologists sometimes argue that intelligence is a rare development—though in the Darwinian approach it certainly has clear benefit. In addition, particularly in the Cold War period, it was often said that technological civilizations would quickly tend to destroy themselves, but now it seems more likely that intelligence—once developed—will tend to survive indefinitely, at least in machine form.

It is obviously difficult to guess the possible motivations of extraterrestrials, but one might expect that—just as with humans—different extraterrestrials would tend to do different things, so that at least some would choose to send out signals if not spacecraft. Out of about 6 billion humans, however, it is notable that only extremely few choose, say, to explore life in the depths of the oceans—though perhaps this is just because technology has not yet made it easy to do. In human history a key motivator for exploration has been trade. But trade requires that there be things of value to exchange; yet it is not clear that with sufficiently advanced technology there would be. For if the fundamental theory of physics is known, then everything about what is possible in our universe can in principle be worked out purely by a computation. Often irreducible work will be required, which one might imagine it would be worthwhile to trade. But as a practical matter, it seems likely that there will be vastly more room to do more extensive computations by using smaller components than by trading and collaborating with even millions of other civilizations. (It is notable that just a couple of decades ago, it was usually assumed that extraterrestrials would inevitably want to use large amounts of energy, and so would eventually for example tap all the output of a star. But seeing the increasing emphasis on information rather than mechanical work in human affairs this now seems much less clear.)

Extrapolating from our development, one might expect that most extraterrestrials would be something like immortal disembodied minds. And what such entities might do has to some extent been considered in the context of the notion of heaven in theology and art. And it is perhaps notable that while such activities as music and thought are often discussed, exploration essentially never is.

▪ **Physics as intelligence.** From the point of view of traditional thinking about intelligence in the universe it might seem like an extremely bizarre possibility that perhaps intelligence could exist at a very small scale, and in effect have spread throughout the universe, building as an artifact everything we see. But at least with a broad interpretation of intelligence this is at some level exactly what the Principle of Computational Equivalence suggests has actually happened. For it implies that even at the smallest scales the laws of physics will show the same computational sophistication that we normally associate with intelligence. So in some sense this

supports the theological notion that there might be a kind of intelligence that permeates our universe. (See page 1195.)

Implications for Technology

▪ **Covering technology.** In writing this book I have tried to achieve some level of completeness in covering the obvious scientific implications of my ideas. But to cover technological implications at anything like the same level would require at least as long a book again. And in my experience many of the intellectually most interesting aspects of technology emerge only when one actually tries to build technology for real—and they are often in a sense best captured by the technology itself rather than by a book about it.

▪ **Page 840 · Applications of randomness.** Random drawing of lots has been used throughout recorded history as an unbiased way to distribute risks or rewards. Also common have been games of chance (see page 968). Randomness is in general a convenient way to allow local decisions to be made while maintaining overall averages. In biological organisms it is used in determining sex of offspring, as well as in achieving uniform sampling, say in foraging for food. (Especially in antiquity, all sorts of seemingly random phenomena have been used as a basis for fortune telling.)

The notion of taking random samples as a basis for making unbiased deductions has been common since the early 1900s, notably in polling and market research. And in the past few decades explicit randomization has become common as a way of avoiding bias in cases such as clinical trials of drugs.

In the late 1800s it was noted in recreational mathematics that one could find the value of π by looking at randomly dropped needles. In the early 1900s devices based on randomness were built to illustrate statistics and probability (see page 312), and were used for example to find the form of the Student *t*-distribution. With the advent of digital computers in the mid-1940s Monte Carlo methods (see page 968) were introduced, initially as a way to approximate processes like neutron diffusion. (Similar ideas had been used in 1901 by Kelvin to study the Boltzmann equation.) Such methods became increasingly popular, especially for simulating systems like telephone networks and particle detectors that have many heterogeneous elements—as well as in statistical physics. In the 1970s they also became widely used for high-dimensional numerical integration, notably for Feynman diagram evaluation in quantum electrodynamics. But eventually it was realized that quasi-Monte Carlo methods based on simple sequences could normally do better than ones based on pure randomness (see page 1085).

A convenient way to tell whether expressions are equal is to evaluate them with random numerical values for variables. (Care must be taken with branch cuts and bounding intervals for inexact numbers.) In the late 1970s it was noted that by evaluating `PowerMod[a, n - 1, n] == 1` for several random integers `a` one can with high probability quickly deduce `PrimeQ[n]`. (In the 1960s it had been noted that one can factor polynomials by filling in random integers for variables and factoring the resulting numbers.) And in the 1980s many such randomized algorithms were invented, but by the mid-1990s it was realized that most did not require any kind of true randomness, and could readily be derandomized and made more predictable. (See page 1085.)

There are all sorts of situations where in the absence of anything better it is good to use randomness. Thus, for example, many exploratory searches in this book were done randomly. And in testing large hardware and software systems random inputs are often used.

Randomness is a common way of avoiding pathological cases and deadlocks. (It requires no communication between components so is convenient in parallel systems.) Examples include message routing in networks, retransmission times after ethernet collisions, partitionings for sorting algorithms, and avoiding getting stuck in minimization procedures like simulated annealing. (See page 347.) As on page 333, it is common for randomness to add robustness—as for example in cellular automaton fluids, or in saccadic eye movements in biology.

In cryptography randomness is used to make messages look typical of all possibilities (see page 598). It is also used in roughly the same way in hashing (see page 622). Such randomness must be repeatable. But for cryptographic keys it should not be. And the same is true when one picks unique IDs, say to keep track of repeat web transactions with a low probability of collisions. Randomness is in effect also used in a similar way in the shotgun method for DNA sequencing, as well as in creating radar pulses that are difficult to forge. (In biological organisms random diversity in faces and voices may perhaps have developed for similar reasons.)

The unpredictability of randomness is often useful, say for animals or military vehicles avoiding predators (see page 1105). Such unpredictability can also be used in simulating human or natural processes, say for computer graphics, videogames, or mock handwriting. Random patterns are often used as a way to hide regularities—as in camouflage, security envelopes, and many forms of texturing and distressing. (See page 1077.)

In the past, randomness was usually viewed as a thing to be avoided. But with the spread of computers and consumer

electronics that normally operate predictably, it has become increasingly popular as an option.

Microscopic randomness is implicitly used whenever there is dissipation or friction in a system, and generally it adds robustness to the behavior that occurs in systems.

▪ **Page 841 · Self-assembly.** Given elements (such as pieces of molecules) that fit together only when certain specified constraints are satisfied it is fairly straightforward to force, say, cellular automaton patterns to be generated, as on page 221. (Notable examples of such self-assembly occur for instance in spherical viruses.)

▪ **Page 841 · Nanotechnology.** Popular since the late 1980s, especially through the work of Eric Drexler, nanotechnology has mostly involved investigation of several approaches to making essentially mechanical devices out of small numbers of atoms. One approach extrapolates chip technology, and studies placing atoms individually on solid surfaces using for example scanning probe microscopy. Another extrapolates chemical synthesis—particularly of fullerenes—and considers large molecules made for example out of carbon atoms. And another involves for example setting up fragments of DNA to try to force particular patterns of self-assembly. Most likely it will eventually be possible to have a single universal system that can manufacture almost any rigid atomic-scale structure on the basis of some kind of program. (Ribosomes in biological cells already construct arbitrary proteins from DNA sequences, but ordinary protein shapes are usually difficult to predict.) Existing work has tended to concentrate on trying to make rather elaborate components suitable for building miniature versions of familiar machines. The discoveries in this book imply however that there are much simpler components that can also be used to set up systems that have behavior with essentially any degree of sophistication. Such systems can either have the kind of chemical and mechanical character most often considered in nanotechnology, or can be primarily electronic, for example along the lines of so-called quantum-dot cellular automata. Over the next several decades applications of nanotechnology will no doubt include much higher-capacity computers, active materials of various kinds, and cellular-scale biomedical devices.

▪ **Page 842 · Searching for technology.** Many inventions are made by pure ingenuity (sometimes aided by mathematical calculation) or by mimicking processes that go on in nature. But there are also cases where systematic searches are done. Notable examples were the testing of thousands of materials as candidate electric light bulb filaments by Thomas Edison in 1879, and the testing of 606 substances for chemotherapy by Paul Ehrlich in 1910. For at least fifty years it has now been quite routine to test many hundreds or thousands of substances in looking, say, for catalysts or drugs with particular functions. (Other kinds of systematic searches done include ones for metal alloys, cooking recipes and plant hybrids.) Starting in the late 1980s the methods of combinatorial chemistry (see note below) began to make it possible to do biochemical tests on arrays of millions of related substances. And by the late 1990s, similar ideas were being used for example in materials science: in a typical case an array of different combinations of substances is made by successively spraying through an appropriate sequence of masks, with some physical or chemical test then applied to all the samples.

In the late 1950s maximal length shift register sequences (page 1084) and some error-correcting codes (page 1101) were found by systematic searches of possible polynomials. Most subsequent codes, however, have been found by explicit mathematical constructions. Optimal circuit blocks for operations such as addition and sorting (see page 1142) have occasionally been found by searches, but are more often found by explicit construction, progressive improvement or systematic logic minimization (see page 1097). In some compilers searches are occasionally done for optimal sequences of instructions to implement particular simple functions. And in recent years—notably in the building of *Mathematica*—optimal algorithms for operations such as function evaluation and numerical integration have sometimes been found through searches. In addition, my 1984 identification of rule 30 as a randomness generator was the result of a small-scale systematic search.

Particularly since the 1970s, many systematic methods have been tried for optimizing engineering designs by computer. Usually they are based on iterative improvement rather than systematic search. Some rely on linear programming or gradient descent. Others use methods such as simulated annealing, neural networks and genetic algorithms. But as discussed on page 342, except in very simple cases, the results are usually far from any absolute optimum. (Plant and animal breeding can be viewed as a simple form of randomized search done since the dawn of civilization.)

▪ **Page 843 · Methodology in this book.** Much of what is presented in this book comes from systematic enumeration of all possible systems of particular types. However, sometimes I have done large searches for systems (see e.g. page 112). And especially in Chapter 11 I have occasionally explicitly constructed systems that show particular features.

▪ **Chemistry.** Chemical compounds are a little like cellular automata and other kinds of programs. For even though

the basic physical laws relevant to chemical compounds have been known since the early 1900s, it remains extremely difficult to predict the actual properties of a given compound. And I suspect that the ultimate reason for this—just as in the case of simple programs—is computational irreducibility.

For a single molecule, the minimum energy configuration can presumably always be found by a limited amount of computational work—though potentially increasing rapidly with the number of atoms. But if one allows progressively more molecules computational irreducibility can make it take progressively more computational work to see what will happen. And much as in determining whether constraints like those on page 213 can be satisfied for an infinite region, it can take an infinite amount of computational work to determine bulk properties of an infinite collection of molecules. Thus in practice it has typically been difficult to predict for example boiling and particularly melting points (see note below). So this means in the end that most of chemistry must be based on facts determined experimentally about specific compounds that happen to have been studied.

There are currently about 10 million compounds listed in standard chemical databases. Of these, most were first identified as extracts from biological or other natural systems. In trying to discover compounds that might be useful say as drugs the traditional approach was to search large libraries of compounds, then to study variations on those that seemed promising. But in the 1980s it began to be popular to try so-called rational design in which molecules were created that could at least to some extent specifically be computed to have relevant shapes and chemical functions. Then in the 1990s so-called combinatorial chemistry became popular, in which—somewhat in imitation of the immune system—large numbers of possible compounds were created by successively adding at random several different possible amino acids or other units. But although it will presumably change in the future it remained true in 2001 that half of all drugs in use are derived from just 32 families of compounds.

Doing a synthesis of a chemical is much like constructing a network by applying a specified sequence of transformations. And just like for multiway systems it is presumably in principle undecidable whether a given set of possible transformations can ever be combined to yield a particular chemical. Yet ever since the 1960s there have been computer systems like LHASA that try to find synthesis pathways automatically. But perhaps because they lack even the analog of modern automated theorem-proving methods, such systems have never in practice been extremely successful.

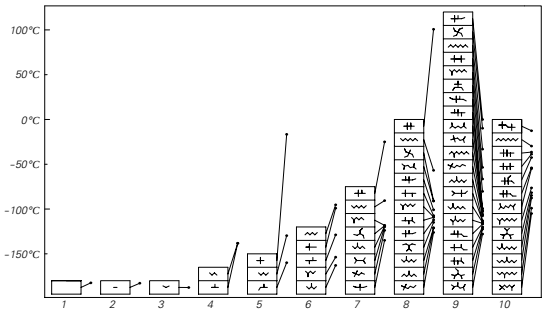
■ **Interesting chemicals.** The standard IUPAC system for chemical nomenclature assigns a name to essentially any possible compound. But even among hydrocarbons with fairly few atoms not all have ever been considered interesting enough to list in standard chemical databases. Thus for example the following compares the total number of conceivable alkanes (paraffins) to the number actually listed in the 2001 standard Beilstein database:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
total	1	1	1	2	3	5	9	18	35	75	159	355	802	1858	4347	10359
listed	1	1	1	2	3	5	9	18	35	75	68	108	60	60	41	62

Any tree with up to 4 connections at each node can in principle correspond to an alkane with chemical formula C_nH_{2n+2}. The total number of such trees—studied since 1875—increases roughly like $2.79^n\, n^{-5/2}$. If every node has say 4 connections, then eventually one gets dendrimers that cannot realistically be constructed in 3D. But long before this happens one runs into many alkanes that presumably exist, but apparently have never explicitly been studied. The small unbranched ones (methane, ethane, propane, butane, pentane, etc.) are all well known, but ones with more complicated branching are decreasingly known. In coal and petroleum a continuous range of alkanes occur. Branched octanes are used to reduce knocking in car engines. Biological systems contain many specific alkanes—often quite large—that happen to be produced through chemical pathways in biological cells. (The $n = 11$ and $n = 13$ unbranched alkanes are for example known to serve as ant pheromones.)

In general the main way large molecules have traditionally ended up being considered chemically interesting is if they occur in biological systems—or mimic ones that do. Since the 1980s, however, molecules such as the fullerenes that instead have specific regular geometrical shapes have also begun to be considered interesting.

■ **Alkane properties.** The picture on the facing page shows melting points measured for alkanes. (Note that even when alkanes are listed in chemical databases—as discussed above—their melting points may not be given.) Unbranched alkanes yield melting points that increase smoothly for n even and for n odd. Highly symmetrical branched alkanes tend to have high melting points, presumably because they pack well in space. No reliable general method for predicting melting points is however known (see note above), and in fact for large n alkanes tend to form jellies with no clear notion of melting.



Things appear somewhat simpler with boiling points, and as noticed by Harry Wiener in 1947 (and increasingly discussed since the 1970s) these tend to be well fit as being linearly proportional to the so-called topological index given by the sum of the smallest numbers of connections visited in getting between all pairs of carbon atoms in an alkane molecule.

▪ **Page 843 · Components for technology.** The Principle of Computational Equivalence suggests that a vast range of systems in nature can all ultimately be used to make computers. But it is remarkable to what extent even the components of present-day computer systems involve elements of nature originally studied for quite different reasons. Examples include electricity, semiconductors (used for chips), ferrites (used for magnetic storage), liquid crystals (used for displays), piezoelectricity (used for microphones), total internal reflection (used for optical fibers), stimulated emission (used for lasers) and photoconductivity (used for xerographic printing).

▪ **Future technology.** The purposes technology should serve inevitably change as human civilization develops. But at least in the immediate future many of these purposes will tend to relate to the current character of our bodies and minds. For certainly technology must interface with these. But presumably as time progresses it will tend to become more integrated, with systems that we have created eventually being able to fit quite interchangeably into our usual biological or mental setup. At first most such systems will probably tend either to be based on standard engineering, or to be quite direct emulations of human components that we see. But particularly by using the ideas and methods of this book I suspect that significant progressive enhancements will be possible. And probably there will be many features that are actually quite easy to take far beyond the originals. One example is memory and the recall of history. Human memory is in many ways quite impressive. Yet for ordinary physical objects we are used to the idea that they remember little of their history, for at a macroscopic level we tend to see only the coarsest traces. But at a microscopic scale something like the surface of a solid has in at least some form remarkably detailed information about its history. And as technological systems get smaller it should become possible to read and manipulate this. And much as in the discussion at the end of Chapter 10 the ability to interact at such a level will yield quite different experiences, which in turn will tend to suggest different purposes to pursue with technology.

Historical Perspectives

▪ **Page 844 · Human uniqueness.** The idea that there is something unique and special about humans has deep roots in Judeo-Christian tradition—and despite some dilution from science remains a standard tenet of Western thought today. Eastern religions have however normally tended to take a different view, and to consider humans as just one of many elements that make up the universe as a whole. (See note below.)

▪ **Page 845 · Animism.** Belief in animism remains strong in perhaps several hundred million indigenous people around the world. In its typical form, it involves not only explaining natural phenomena by analogy to human behavior but also assuming that they can be influenced as humans might be, say by offerings or worship. (See also page 1177.)

Particularly since Edward Tylor in 1871 animism has often been thought of as the earliest identifiable form of religion. Polytheism is then assumed to arise when the idea of localized spirits associated with individual natural objects is generalized to the idea of gods associated with types of objects or concepts (as for example in many Roman beliefs). Following their rejection in favor of monotheism by Judaism—and later Christianity and Islam—such ideas have however tended to be considered primitive and pagan. In Europe through the Middle Ages there nevertheless remained widespread belief in animistic kinds of explanations. And even today some Western superstitions center on animism, as do rituals in countries like Japan. Animism is also a key element of the New Age movement of the 1960s, as well as of such ideas as the Gaia Hypothesis.

Particularly since the work of Jean Piaget in the 1940s, young children are often said to go through a phase of animism, in which they interact with complex objects much as if they were alive and human.

▪ **Page 845 · Universe as intelligent.** Whether or not something like thinking can be attributed to the universe has long been discussed in philosophy and theology. Theism and the standard Western religions generally attribute thinking to a

person-like God who governs the universe but is separate from it. Deism emphasizes that God can govern the universe only according to natural laws—but whether or not this involves thinking is unclear. Pantheism generally identifies the universe and God. In its typical religious form in Eastern metaphysics—as well as in philosophical idealism—the contents of the universe are identified quite directly with the thoughts of God. In scientific pantheism the abstract order of the universe is identified with God (often termed "Nature's God" or "Spinoza's God"), but whether this means that thinking is involved in the operation of the universe is not clear. (See also pages 822 and 1191.)

▪ **Non-Western thinking.** Some of my conclusions in this book may seem to resonate with ideas of Eastern thinking. For example, what I say about the fundamental similarity of human thinking to other processes in nature may seem to fit with Buddhism. And what I say about the irreducibility of processes in nature to short formal rules may seem to fit with Taoism. Like essentially all forms of science, however, what I do in this book is done in a rational tradition—with limited relation to the more mystical traditions of Eastern thinking.

▪ **Aphorisms.** Particularly from ancient and more fragmentary texts aphorisms have survived that may sometimes seem at least vaguely related to this book. (An example from the pre-Socratics is "everything is full of gods".) But typically it is impossible to see with any definiteness what such aphorisms might really have been intended to mean.

▪ **Postmodernism.** Since the mid-1960s postmodernism has argued that science must have fundamental limitations, based on its general belief that any single abstract system must somehow be as limited—and as arbitrary in its conclusions—as the context in which it is set up. My work supports the notion that—despite implicit assumptions made especially in the physical sciences—context can in fact be crucial to the choice of subject matter and interpretation of results in science (see e.g. page 1105). But the Principle of Computational Equivalence suggests at some level a remarkable uniformity among systems, that allows all sorts of general scientific statements to be made without dependence on context. It so happens that some of these statements then imply intrinsic general limitations on science—but even the very fact that such statements can be made is in a sense an example of successful generality in science that goes against the conclusions of postmodernism. (See also page 1131.)

▪ **Microcosm.** The notion that a human mind might somehow be analogous to the whole universe was discussed by Plato and others in antiquity, and known in the Middle Ages. But it was normally assumed that this was something fairly unique to the human mind—and nothing with the generality of the Principle of Computational Equivalence was ever imagined.

▪ **Human future.** The Principle of Computational Equivalence and the results of this book at first suggest a rather bleak view of the end point of the development of technology. As I argued in Chapter 10 computers will presumably be able to emulate human thinking. And particularly using the methods of this book one will be able to use progressively smaller physical components as elements of computers. So before too long it will no doubt be possible to implement all the processes of thinking that go on in a single human—or even in billions of humans—in a fairly small piece of material. Each piece of human thinking will then correspond to some microscopic pattern of changes in the atoms of the material. In the past one might have assumed that these changes would somehow show fundamental evidence of representing sophisticated human thinking. But the Principle of Computational Equivalence implies that many ordinary physical processes are computationally just as sophisticated as human thinking. And this means that the pattern of microscopic changes produced by such processes can at some level be just as sophisticated as those corresponding to human thinking. So given, say, an ordinary piece of rock in which there is all sorts of complicated electron motion this may in a fundamental sense be doing no less than some system of the future constructed with nanotechnology to implement operations of human thinking. And while at first this might seem to suggest that the rich history of biology, civilization and technology needed to reach this point would somehow be wasted, what I believe instead is that this just highlights the extent to which such history is what is ultimately the defining feature of the human condition.

▪ **Philosophical implications.** The Principle of Computational Equivalence has implications for many issues long discussed in the field of philosophy. Most important are probably those in epistemology (theory of knowledge). In the past, it has usually been assumed that if we could only build up in our minds an adequate model of the world, then we would immediately know whatever we want about the world. But the Principle of Computational Equivalence now implies that even given a model it may be irreducibly difficult to work out its consequences. In effect, computational irreducibility introduces a new kind of limit to knowledge. And it implies that one needs a criterion more sophisticated than immediate predictability to assess a scientific theory—since when computational irreducibility is present this will inevitably be limited. In the past, it has sometimes been assumed that truths that can be deduced purely by operations like those in logic must somehow always be trivial. But computational

irreducibility implies that in general they are not. Indeed it implies that even once the basic laws are known there are still an endless series of questions that are worth investigating in science. It is often assumed that one cannot learn much about the world just by studying purely formal systems—and that one has to rely on empirical input. But the Principle of Computational Equivalence implies that at some level there are inevitably common features across both abstract and natural systems. In ontology (theory of being) the Principle of Computational Equivalence implies that special components are vastly less necessary than might have been thought. For it shows that all sorts of sophisticated characteristics can emerge from the very same kinds of simple components. (My discussion of fundamental physics in Chapter 9 also suggests that no separate entities beyond simple rules are needed to capture space, time or matter.) Arguments in several areas of philosophy involve in effect considering fundamentally different intelligences. But the Principle of Computational Equivalence implies that in fact above a certain threshold there is an ultimate equivalence between possible intelligences. In addition, the Principle of Computational Equivalence implies that all sorts of systems in nature and elsewhere will inevitably exhibit features that in the past have been considered unique to intelligence—and this has consequences for the mind-body problem, the question of free will, and recognition of other minds. It has often been thought that traditional logic—and to some extent mathematics—are somehow fundamentally special and provide in a sense unique foundations. But the Principle of Computational Equivalence implies that in fact there are a huge range of other formal systems, equivalent in their ultimate richness, but different in their details, and in the questions to which they naturally lead. In philosophy of science the Principle of Computational Equivalence forces a new methodology based on formal experiments—that is ultimately the foundation for the whole new kind of science that I describe in this book.