The Lifebox, the Seashell, and the Soul

The Lifebox, the Seashell, and the Soul

WHAT GNARLY COMPUTATION
TAUGHT ME ABOUT ULTIMATE REALITY,
THE MEANING OF LIFE, AND HOW TO BE HAPPY

Rudy Rucker

THUNDER'S MOUTH PRESS
NEW YORK

THE LIFEBOX, THE SEASHELL, AND THE SOUL
WHAT GNARLY COMPUTATION TAUGHT ME ABOUT ULTIMATE REALITY, THE MEANING OF LIFE, AND HOW TO BE HAPPY

Published by
Thunder's Mouth Press
An Imprint of Avalon Publishing Group Inc.
245 West 17th St., 11th Floor
New York, NY 10011

First printing October 2005

Library of Congress Cataloging-in-Publication Data is available.

ISBN 1-56025-722-9

9 8 7 6 5 4 3 2 1

Illustrations by Isabel Rucker

Book design by Maria E. Torres

Printed in the United States of America
Distributed by Publishers Group West

Contents

Preface

As a teenager in 1961, I imagined that I'd like to become a philosopher. I recall agreeing with my best friend, Niles Schoening, that what we'd most like to do would be to get college degrees in philosophy and spend the rest of our lives as bums, talking about the meaning of life.

As it turned out, I ended up getting a Ph.D. in mathematical logic. And instead of becoming a bum, I found work as a professor and as a writer of popular science and science fiction. I did keep talking about the meaning of life, though, to the point of publishing three somewhat philosophical books about mathematics: *The Fourth Dimension, Infinity and the Mind,* and *Mind Tools*.

In the mid-1980s I sensed something new in the air. Computers were ushering in an era of experimental mathematics. Fractals, chaos, cellular automata, artificial life! And when I interviewed Stephen Wolfram for a magazine article, my fate was sealed. I moved to Silicon Valley, retooled, and became a computer science professor at San Jose State University, also doing some work as a programmer for the computer graphics company Autodesk.

Back when I was contemplating my big switch to computer science, my old friend Gregory Gibson said something encouraging. "Imagine if William Blake had worked in a textile mill. What might he have written then?"

Initially, I thought this might be a quick foray. Get in, figure out what's happening, get out, and write my book on computers and reality. But somewhere along the way I went native on the story. I all but forgot my mission.

I spent twenty years in the dark satanic mills of Silicon Valley. I'm covered in a thick lint of bytes and computer code. And now I'm stepping into the light to tell you what I learned among the machines.

I'm grateful to the Royal Flemish Academy of Belgium for Science and the Arts for having funded a stay in Brussels in the fall of 2002. I taught a course on the philosophy of computer science at the University of Leuven, writing some material for this book in the process. I gave my classroom handouts the not-quite-serious title, "Early Geek Philosophy," telling the students that my precursors might come to be known as the pre-Rucratic geek philosophers!

Many thanks also to the people with whom I've had conversations and/or e-mail exchanges about the book's topics. These include: Scott Aaronson, Ralph Abraham, Mark van Atten, Michael Beeson, Charles Bennett, Kovas Boguta, Jason Cawley, Leon Horsten, Loren Means, Jon Pearce, Chris Pollett, Richard Shore, Brian Silverman, John Walker, Ken Wharton, and Stephen Wolfram. Some of these friends even did me the favor of reading an early draft and suggesting corrections. Errors that remain are my own responsibility.

Thanks also to my computer science students at San Jose State University; my programs that illustrate this book were developed with them in mind, and sometimes they even helped me write them.

And special thanks to my wife, Sylvia, for all the wonderful things outside the ambit of the buzzing machines.

Rudy Rucker
Los Gatos, California
March 22, 2005

THOUGHT EXPERIMENT ONE: LUCKY NUMBER

Just for fun, I've written a short-short story to introduce each chapter of The Lifebox, the Seashell, and the Soul. *You might think of these as thought experiments, or as exploratory expeditions into the further reaches of my book's themes.*

The first Sunday in October, Doug Cardano drove in for an extra day's work at Giga Games. Crunch time. The nimrods in marketing had committed to shipping a virtual reality golf game in time for the holiday season. NuGolf. It was supposed to have five eighteen-hole courses, all of them new, all of them landscaped by Doug.

He exited Route 101 and crossed the low overpass over the train tracks, heading toward the gleaming Giga Games complex beside the San Francisco Bay. A long freight train was passing. Growing up, Doug had always liked trains; in fact, he'd dreamed of being a hobo. Or an artist for a game company. He hadn't known about crunch time.

Just to postpone the start of his long, beige workday, he pulled over and got out to watch the cars clank past: boxcars, tankers, reefers, flatcars. Many of them bore graffiti. Doug lit a cigarette, his first of the day, always the best one, and spotted a row of twelve spray-painted numbers on a dusty red boxcar, the digits arranged in pairs.

11 35 17 03 25 14

SuperLotto, thought Doug, and wrote them on his cardboard box of cigarettes. Five numbers between one and forty-seven, and one number between one and twenty-seven.

Next stop was the minimarket down the road. Even though Doug knew the odds were bogus, he'd been buying a lot of SuperLotto tickets lately. The grand prize was hella big. If he won, he'd never have to crunch again.

The rest of the team trickled into the office about the same time as Doug. A new bug had broken one of the overnight builds, and Van the lead coder had to fix that. Meanwhile Doug got down to the trees and bushes for course number four.

Since the player could mouse all around the NuGolf world and even

wander into the rough, Doug couldn't use background bitmaps. He had to create three-dimensional models of the plants. NuGolf was meant to be wacky and fantastic, so he had a lot of leeway: on the first course he'd used cartoony saguaro cactuses, he'd set the second links underwater with sea fans and kelp, the third had been on "Venus" with man-eating plants, and for the fourth, which he was starting today—well, he wasn't sure what to do.

He had a vague plan of trying to get some inspirations from BlobScape, a three-dimensional cellular automata package he'd found on the Web. Cellular automata grew organic-looking objects on the fly. Depending what number you seeded BlobScape with, it could grow almost anything. The guy who'd written BlobScape claimed that theoretically the computation could simulate the whole universe, if only you gave it the right seed.

When Doug started up BlobScape today, it was in a lava lamp mode, with big wobbly droplets drifting around. A click of the Randomize button turned the blobs into mushroom caps, pulsing through the simulation space like jellyfish. Another click produced interlocking pyramids a bit like trees, but not pretty enough to use.

Doug pressed the Rule button so he could enter some code numbers of his own. He'd done this a few times before; every now and then his numbers would make something really cool. It reminded him of the Magic Rocks kit he'd had as boy, where the right kind of gray pebble in a glass of liquid could grow green and purple stalagmites. Maybe today was his lucky day. Come to think of it, his SuperLotto ticket happened to be lying on his desk, so, what the hey, he entered 113517032514.

Bingo. The block of simulated space misted over, churned, and congealed into—a primeval jungle inhabited by dinosaurs. And it kept going from there. Apemen moved from the trees into caves. Egyptians built the Sphinx and the pyramids. A mob crucified Christ. Galileo dropped two balls off the Leaning Tower of Pisa. Soldiers massacred the Indians of the Great Plains. Flappers and bootleggers danced the jitterbug. Hippies handed out daisies. Computers multiplied like bacilli.

Doug had keyed in the Holy Grail, the one true rule, the code number for the universe. Sitting there grinning, it occurred to him that if you wrote those twelve lucky digits in reverse order they'd work as a phone number plus extension. (415) 230-7135 x11.

The number seemed exceedingly familiar, but without stopping to think he went ahead and dialed it.

His own voice answered.

"Game over."

The phone in Doug's hand turned into pixels. He and the phone and the universe dissolved.

CHAPTER ONE

Computation Everywhere

1.1: Universal Automatism

The Lifebox, the Seashell, and the Soul is about computation—taken in the broadest possible sense. You can usefully regard all sorts of things as computations: plants and animals, political movements, the weather, your personal shifts of mood. Computations are everywhere, once you begin to look at things in a certain way. At the very least, computation is a metaphor with an exceedingly wide range of applicability.

You may feel a twinge of déjà vu. After all, it hasn't been so long since guys like me were telling you everything is information, and aren't information and computation pretty much the same? No—information is static, but computation is dynamic. Computations transform information.

An example. Even if we could find a complete and correct explanation of our world's physics, this would only be a static piece of information—perhaps an initial condition and a set of rules. The interesting things happen as the consequences of the laws unfold. The unfolding process is a computation carried out by the world itself.

My iconoclastic and headstrong friend Stephen Wolfram goes so far as to say that the world is nothing more than a computation. In Wolfram's words, "It is possible to view every process that occurs in nature or elsewhere as a computation." I call this view *universal automatism.*[1]

I'm not sure if I subscribe to universal automatism or not. One reason I'm writing this book is to see where universal automatism leads.

Does my willingness to entertain universal automatism mean that I'm a humorless nerd who wants everything to fit into the beige box of a personal

computer (PC)? No way. I know a lot about PCs, yes, but familiarity breeds contempt. Although I've gotten very skilled at crafting C++ and Java code for the classes I've taught, I don't think I'd much mind if I never had to write a computer program again. Writing books is a lot more fun—programming is simply too brittle a medium. If I leave out, say, a semicolon, a program might not run at all. Literature isn't like that.

And, just like most people, I have a deep-seated conviction that I myself am something richer than any mere calculation. I love to get away from my flickering monitor screen and be out in nature—roaming the woods, bicycling down the flowery California streets, walking on a beach, or just sitting in my backyard watching the wind move the leaves on the trees. Crows, ants, dogs, protozoa, other people—life is what matters, not some crappy buzzing boxes that are broken half the time.

No, no, I'm not on the side of machines.

But then why am I writing this long book about computation? I guess I've put in so much time with personal computers that I'd like to take this one last shot at figuring out what I've learned from them. To trace out their meanings once and for all.

My original training was in mathematics—thirty years ago I got a Ph.D. in set theory and mathematical logic. In the 1970s I even got to meet Kurt Gödel a few times. The king of the logicians. Gödel once told me, "The a priori is very powerful." By this he meant that pure logic can take you farther than you might believe possible.

As well as logic, I've got a lot of experimental input to work with. Wolfram, whom I first met in the 1980s, has done a king-hell job of combing through vast seas of possible computations, getting a handle on the kinds of phenomena that can occur. With Wolfram's discoveries, and with my own experiences as a logician, a programmer, and a computer science professor—well, I'm hoping I can make a little progress here.

But let me repeat: I'm not a big fan of machines.

Being a good Californian, I practice yoga nearly every day. It counteracts the strain on my aging frame from the huge amount of keyboarding and mouse-clicking that I do. It's interesting how good it feels to stop worrying about my daily plans and focus on nothing more than my breath and my muscles. Can computation theory tell me anything about yoga?

Years ago—this would have been the glorious summer of 1969—I had a

vision of God. The White Light, the Supreme Being, all around me, talking to me. "I'm always here, Rudy," said God. "I love you." These days I don't see God. But when I remember to try, I can still feel something like a divine presence. I don't think God's a computation. But exactly why not?

I'm a novelist, too, mostly science fiction, and I presume to think of my work as literature. Compared to a work of literature, a computer program is puny excrescence, a petty game played by the rules of blind machines, a dreary slog through the mud. Literature glides on beautiful wings. But maybe, looked in a certain light, literature is a human form of computation.

When I open my heart to universal automatism, I can see that it's not as farfetched as it sounds. The key fact is that, far from being dry and dull, computations can generate physical, biological, and psychological phenomena of great beauty. Maybe a weird explanation is better than no explanation at all. Might it be that, by analyzing the notion of computation, I can finally understand what it means to be conscious? I'm prepared to follow the argument wherever it goes. If it turns out that universal automatism is right, and I really am a computation, then at least I'll know a little more about what *kind* of computation.

In planning this intellectual journey, I've settled on a particular tactic and an overall strategy. My tactic is Hegelian dialectic, and my strategy is what we might call the stairway to heaven.

The dialectic tactic relates to the book's title: *The Lifebox, the Seashell, and the Soul*. The title represents a triad: the lifebox is the thesis, the soul is the antithesis, and the seashell is the synthesis. In the style of my great-great-great-grandfather Georg Wilhelm Hegel, my tactic will be to use my selected triad over and over.

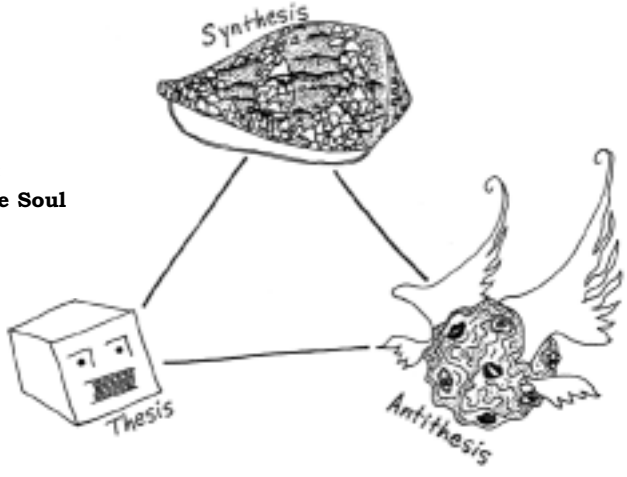


Figure 1: A Dialectic Triad: the Lifebox, the Seashell, and the Soul

Lifebox is a word I invented some years ago to describe a hypothetical technological gizmo for preserving a human personality. In my science-fiction tales, a lifebox is a small interactive device to which you tell your life story. It prompts you with questions and organizes the information you give it. As well as words, you can feed in digital images, videos, sound recordings, and the like. It's a bit like an intelligent blog.

Once you get enough information into your lifebox, it becomes something like a simulation of you. Your audience can interact with the stories in the lifebox, interrupting and asking questions. The lifebox begins by automating the retiree's common dream of creating a memoir and ends by creating a simulation of its owner.

Why would you want to make a lifebox? Immortality, ubiquity, omnipotence. You might leave a lifebox behind so your grandchildren and great-grandchildren can know what you were like. You might use your lifebox as a way to introduce yourself to large numbers of people. You might let your lifebox take over some of your less interesting duties, such as answering routine phone calls and e-mail.

A lifebox is a person reduced to a digital database with simple access software. So in my book title, I'm using *Lifebox* as shorthand for the universal automatist *thesis* that everything, even human consciousness, is a computation.

The *antithesis* is the fact that nobody is really going to think that a wised-up cell phone is alive. We all feel we have something that's not captured by any mechanical model—it's what we commonly call the soul.

My *synthesis* is a Wolfram-inspired scheme for breathing life into a lifebox. The living mind has a churning quality, like the eddies in the wake of a rock in a stream—or like the turbulent patterns found in a certain kind of computation called a cellular automaton (CA). These unpredictable yet deterministic computations are found in nature, perhaps most famously on a kind of seashell called the cone shell (see figure 2). It's at least possible that the mind's endless variety is in fact generated by a gnarly computation of this type. If so, the image of the seashell serves to bridge the chasm between lifebox and soul.

So that's my basic triad, and my dialectic tactic will involve repeatedly going through the following three steps: (a) Thetic step: model some real-world phenomenon as a computation. (b) Antithetic step: remark that the actual

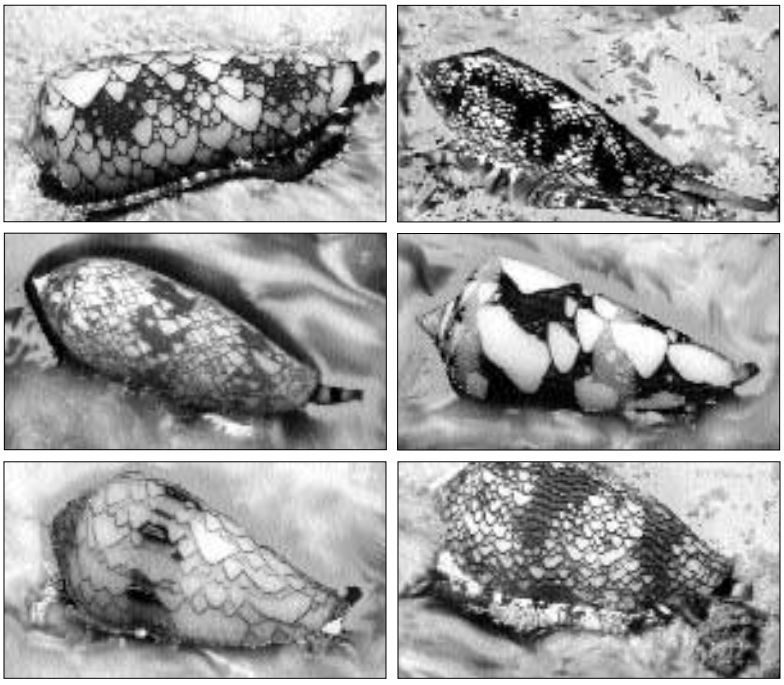


Figure 2: Six Cone Shells

Reading left to right, top to bottom, we have a Conus omaria, Conus auratinus, Conus ammiralis, Conus auricomus, Conus retifer, *and a* Conus textile. *Note that these marine snails have protruding tentacles that are, variously, siphons, mouths, eyes, and proboscises. These so-called tented cones feed upon other mollusks, injecting paralyzing conotoxins into their prey by means of tiny harpoons shot from a tentacle. Shell-collectors have been killed by cone shell stings. Note that the textile cone is in the process of attacking a less-gnarly fellow mollusk. These photos were taken at night by Scott and Jeanette Johnson off the Kwajalein atoll in the Micronesian archipelago.*

world seems juicier and more interesting than a computation. (c) Synthetic step: observe that, given enough time and memory, our proposed computation is in fact capable of generating exceedingly rich and lifelike structures.

When I speak of using a stairway-to-heaven pattern as my overarching strategy for organizing *The Lifebox, the Seashell, and the Soul*, I mean that each chapter treats a yet higher-level way of viewing the world, as suggested in figure 3.

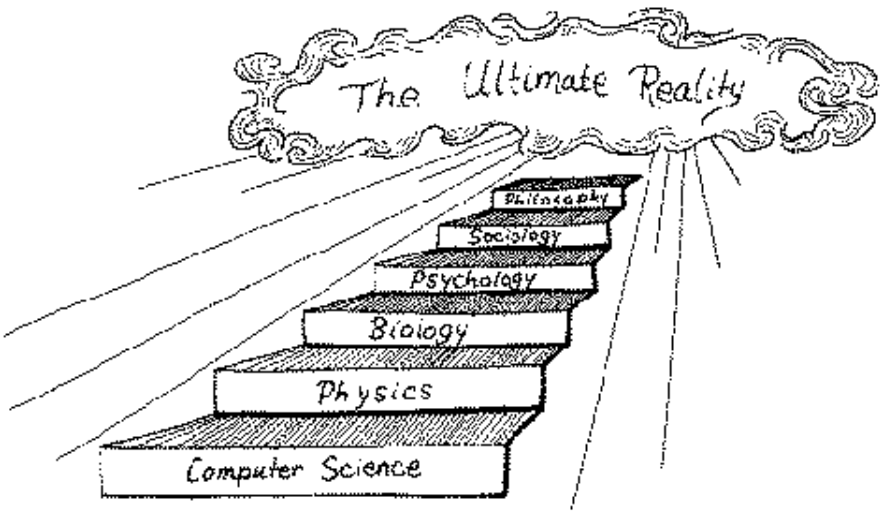


Figure 3: An Intellectual Stairway to Heaven

The stairway to heaven is a traditional style of organizing knowledge. In the Middle Ages it was called *ordo sciendi,* or "the order of knowing." A medieval thinker would of course write "Logic" in place of "Computer Science," and even now someone might say that logic or mathematics would be a more natural starting point than computer science. But in *The Lifebox, the Seashell, and the Soul* I'm arguing that we do best to think of computation itself as fundamental. Under this view, logic and mathematics are invented after the fact to explain the observed patterns of the world. Logic and mathematics become high-level intellectual endeavors that I treat in the context of the sixth chapter, which concerns philosophy.

Looking ahead, my six chapters will be as follows.

- CHAPTER ONE: *Computation Everywhere.* An introduction to the universal automatist view that everything is a computation, exploring the familiar computations done by our machines, and presenting some examples of computational gnarliness.
- CHAPTER TWO: *Our Rich World.* Descriptions of how to view classical, chaotic, and quantum physics in terms of computations.

- CHAPTER THREE: *Life's Lovely Gnarl*. An analysis of life in terms of five kinds of computation: reproduction, morphogenesis, homeostasis, ecology, and evolution, including a discussion of human efforts to create artificial forms of life.
- CHAPTER FOUR: *Enjoying Your Mind.* A detailed presentation of the universal automatist view that we can view the mind as a gnarly computation, showing how this need not contradict one's feeling of being a conscious entity with a soul.
- CHAPTER FIVE: *The Human Hive*. An exploration of the patterns and dynamics of human society from the low to high levels, including discussions of language and culture.
- CHAPTER SIX: *Reality Upgrade*. A philosophical analysis of the possible positions regarding computation and reality, including a description of the classes of computation that are known to exist, and delving further into the philosophical consequences of universal automatism. Concludes with remarks about ultimate reality, the meaning of life, and how to be happy.

And now let's get going on the stairway to heaven's first step.

What do I mean by a computation? Here's a definition that's very minimal—and thus quite generally applicable.

- *Definition*. A *computation* is a process that obeys finitely describable rules.

That's it? Well, if I want to say that all sorts of processes are like computations, it's to be expected that my definition of computation must be fairly simple.

The notion of obeying finitely describable rules really includes two ideas: a computation is utterly *deterministic,* that is, nonrandom, and the rules act as a kind of *recipe* for generating future states of the computation.

Regarding determinism, although computer scientists do sometimes theorize about "probabilistic computations" that are allowed to make utterly random decisions, these aren't really computations in any normal sense of

the word. Our understanding here will be that we're speaking only of computations whose future states are fully determined by their inputs and by their finitely describable rules.

Now I'll talk about the kind of recipe that underlies a finitely describable rule.

You might say that any process at all obeys the recipe of "act like yourself." Does that make for a finitely describable rule? No. The "yourself" in this so-called rule implicitly drags in a full and possibly endless set of information about how "you" will behave in every kind of situation.

Although it may indeed be that every possible process is a kind of computation, I don't want to make it *too* easy to attain this sought-after conclusion. I want "obeying finitely describable rules" to be a real constraint.

A finitely describable collection of rules should be something like a set of behavioral laws, or a program for an electronic digital computer, or a specific scientific theory with its accompanying rules of deduction. What's really intended is that the rules specify what, in any given state, the computational system will do next.

As an aside, I have to warn you that *describable* is a slippery notion. Logicians have established that *describable* can't in fact have a formally precise meaning—otherwise a phrase like the following would be a valid description of a number: "Let the Berry number be the smallest integer that can't be described in less than eighteen words." Now, if that seventeen-word phrase were indeed a legitimate description of a specific Berry number, we'd have the paradox of a seventeen-word phrase describing a number that can't be described in less than eighteen words. So it must be that the phrase really isn't a legitimate description, and the reason must be that *describable* isn't a formally precise word. Therefore, my definition of a computation is imprecise as well. (I wrote at length about this issue in *Infinity and the Mind*.)

So if the notion of a computation is fundamentally imprecise, must we abandon our investigations and sit grumbling in the darkness? No. In this book, I want to think more like a physicist than like a mathematician—more like an experimental scientist and less like a logician. Loosely speaking, we *do* know what it means to have a finite description of a rule for a process. Yes, certain borderline cases will throw us into a philosophical quandaries, but we can cover a lot of ground with our (inherently unformalizable) definition of a computation as a *process that obeys finitely describable rules.*

We can regard a computation as transforming inputs into outputs. Where do the inputs and ouputs live? The inputs and outputs are states of the underlying system that supports the computational process.

Which states are inputs and which are outputs? This is really just a matter of temporal order. If one or more states occurs before a second state, then we speak of the earlier states as inputs that produce the later state. I don't lay much stress on the notion of computations ever coming to a halt, which means that we usually think of an input as producing an endless stream of successive outputs. I'll also allow for additional *interactive* inputs that occur while a computing process is under way.[2]

I want to say a bit about the underlying system that supports a computation. Although it's natural to refer to any such computational system as a *computer,* I need to caution that by "computer" I don't necessarily mean one of our chip-in-a-box machines. Obviously these balky devices are a point of inspiration. But for a universal automatist, essentially *any* system or object is a computer, and any process or action is a computation. To avoid confusion, I'll try to always refer to our day-to-day computing machines as *personal* computers, *electronic* computers, *desktop* computers, or simply *PCs*.

By thinking about PCs we become aware of a number of distinctions. For instance, when speaking of electronic computers, people distinguish between

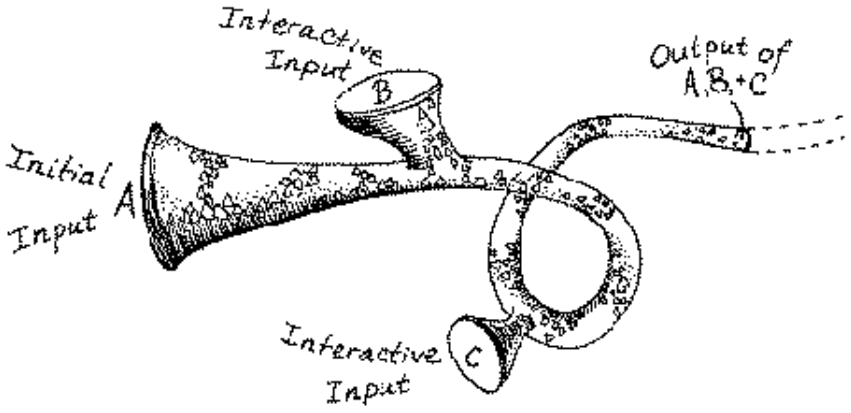


Figure 4: Time Line with Inputs and Outputs

hardware and software. The hardware is the physical contents of the buzzing box that you buy, while the software is the stuff that you get on disks or download from the Web. A finer distinction is possible. The buzzing box comes equipped with its own *low-level software*—which usually includes, for instance, an operating system like Linux, Windows, or the Mac OS. We can distinguish between this low-level software and the specialized *high-level software* that we might want to run—think of, for instance, a word-processing or image-manipulation application. And even higher-level than that are the inputs we feed to our high-level software—think of documents or photo files.

Of course, all of these boundaries are somewhat fuzzy, and the levels are prone to splitting into sublevels. And when we take into account a system's surroundings, new levels appear as shown in figure 5. Suffice it to say that most systems have quite a few levels of rules.

I'll be comparing real-world things to computations for the rest of the book. As a quick example, in a human being, the hardware is like your body and brain, the low-level software is the built-in wiring of your brain and perhaps the effects of your various early experiences, the high-level software is like the skills that you learn, the inputs are your daily experiences, and the outputs are the things that you say and do. Changing the high-level software takes a certain amount of effort, and changing the low-level software is very hard, requiring something on the order of a conversion experience or long-term therapy. There are layers upon layers, and some quirks can be so deeply ingrained that you have to dig down quite far to tweak them.

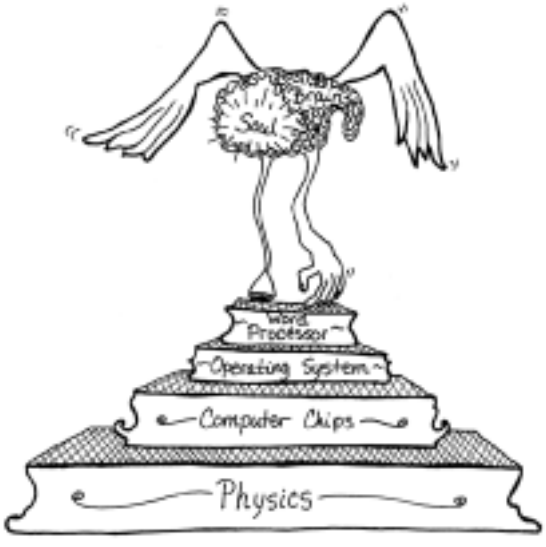


Figure 5: Layers of Computation

Table 1 presents some similar analogies, with a column for seven different kinds of computation, with each column suggesting the hardware, low-level

software, high-level software, inputs, outputs, and possible target detectors for that particular style of computation. (I'll be explaining what I mean by target detectors after the next paragraph. But first a word about tables.)

For me, tables are a tool for thinking. I figure out some column headers and row topics, and then, *wham,* I've got all these nice cells to fill. Let me warn you that you need to take my tables with a grain of salt. They're Procrustean beds. In Greek myth, Procrustes was a bandit masquerading as an inn-keeper. He said he had a wonderful bed that would fit you perfectly, no matter what your size. The catch was, if you were too short for the bed, Procrustes would stretch you on the rack, and if you were too tall, he'd lop off your head or your feet. Filling the cells of a table requires Procrustean fine-tuning—although if it gets out of hand, I usually rethink the row and column categories.

Now I'll tell you about target detectors. This has to do with the issue of when, if ever, I'm going to think of a given computation as being done.

People often suppose that a computation has to "find an answer" and then stop. But our general notion of computation allows for computations that run indefinitely. If you think of your life as a kind of computation, it's quite abundantly clear that there's not going to be a final answer and there won't be anything particularly wonderful about having the computation halt! In other words, we often prefer a computation to yield an ongoing sequence of outputs rather than to attain one final output and turn itself off.

In order to further clarify this point, I'm going to begin speaking a bit symbolically about computations. Throughout the book, I'll normally use the letter *P* to stand for a computation—think of *P* standing for program. If *P* is a computation and In is a state, I write *P*(In) to stand for the indefinitely prolonged computational process that results from starting *P* on In. If Out is another state and *t* is some specific interval of time, I can write *P*(In, *t*) = Out to mean that the computation *P*(In) produces state Out after a time interval *t*.[3]

Even though I'll usually be talking about never-ending computations, in practical uses of computation, there are often situations where we are interested in cases where a computation of the form *P*(In) reaches some targeted state Out and we can then readily perceive that *P*(In) is done. One definition of a computation being *done* (or, as is often said, *halted*) is simply to require that the computation doesn't change any further after some point in time.

Computation system	**Pencil and paper calculations**	**Personal computer**	**The Web**	**Classical physics**	**Quantum physics**	**Organisms**	**Thought**	**Society**
Hardware	Human with pencil	Chip & RAM	Linked computers	Atoms	Reality	Cell	Brain cortex	People, roads, buildings
Low-level software	Plus and times tables, algorithms	Microcode, bios, operating system	File exchange protocols	Physical laws	Schrödinger's wave equation	Ribosomes	Instincts, built-in abilities, reptile brain	Sex, human nature
Application software	Tax form, or math problem	Application code, like Word	Browser, search engine	Design	Experimental apparatus	DNA	Personality, knowledge	Beliefs, news
Input	Numbers	Documents, images, signals from mouse and keyboard	Page address, search request	Forces, energies, objects	State preparation	Food, space, light	Learning, experience	Human actions, natural phenomena
Output	Numbers	Documents, images	Pages, page addresses	Motions, states	Measurements	Motions, actions, reproduction	Utterances, writings, beliefs, emotions	History, social movements
Examples of target detectors	Finishing a calculation	A beep, or something being printed, or the screen settling down, or a wait icon going away	A page finishes downloading, or an upload wait bar goes away	A meter reading settles down, a ball stops moving	A particle triggers a a detector, a person looks at something	Having sex, eating something, dying, moving	Making a decision, saying something aloud, writing something down	Electing someone, choosing the book of the week, deciding to start a war

Table 1: Comparing Various Kinds of Computation Systems

That is, the computational process *P*(In) might reach the state Out and then stay fixed in that state. In this situation we would say that *P*(In) *returns* Out.

The "freezing up" definition of halting is appropriate for certain simple models of computation, such as the abstract devices known as Turing machines. But for more general kinds of computations, freezing up is too narrow a notion of a computation's being done.

For instance, if you want to find out how to get to some street address, you can go on the Web, locate a map site, type in your target address, press Enter, and wait for a few fractions of a second until a little map appears on your screen. The system consisting of your PC plus your Web browser plus the Web itself has carried out a computation and now it's done.

But it wouldn't be at all correct to say that the PC+browser+Web computing system is now frozen in the same state because, for one thing, your Web browser is continually polling your mouse and keyboard to look for new input. And your PC is running all kinds of background processes that never stop. And the other machines that make up the Web certainly haven't stopped just because you found what you were looking for.

When we want to talk about a generalized computational system *P* reaching a target state, we need to have an associated *target detector* computation Is*P*Done, which has two special states that we might as well call True and False. We require that for any output state Out of the system, Is*P*Done(Out) returns either True or False according to whether Out is to be viewed as a target state. Is*P*Done is supposed to be a very simple computation that very quickly enters the final state True or False and remains there.

If we don't explicitly specify the Is*P*Done test, we'll assume that the computation is in a target state if any further updates would leave it in the same state—this is what I meant above by a computation that freezes up. But in the case where *P* is a personal computer or even some naturally occurring system like a pond or a human society, we'll want to use a subtler kind of target detector. How we choose to define a computation's target detector can in fact vary with the kind of inputs we plan to feed to the computation—one can imagine situations where we might say that a pond is in target state when its ripples settle down below some specified level, and a society is in a target state once all the votes in an election have been counted and a new leader has been installed.

So now I've said a bit about computations and the universal automatist notion that they're everywhere, which finishes off my section *1.1: Universal Automatism*. This chapter's remaining sections are as follows.

- *1.2: A New Kind of Science.* In his recent book of this title, Stephen Wolfram divides computations into four classes: those that die out, those that repeat, the messy random-looking ones, and the gnarly ones. For a universal automatist, this suggests new ways to view the whole world.
- *1.3: Reckoning a Sum.* Thinking about how we add numbers with pencil and paper gives a bit of insight into what it means for computations to run at different speeds—an important notion for formulating what it means for a computation to be unpredictable.
- *1.4: Analytical Engines.* I'll give a brief history of how we arrived at the design whereby our electronic computers use programs that are loaded into memory like data. Thanks to this so-called stored program architecture, our PCs are universal in the sense of being able to emulate any other computation.
- *1.5: The Tiniest Brains.* Among the simplest possible computers are the idealized devices known as Turing machines. They're the favorite lab rat of computer philosophers, and they teach us more about universality.
- *1.6: Inside the Beige Box.* A short-as-possible explanation of how our desktop machines work.
- *1.7: Plugged In.* The Internet is a distributed networked computation quite unlike the computations inside a PC.
- *1.8: Flickercladding.* The parallel computations known as cellular automata make beautiful patterns, are good models for physics, and have served as a main source of inspiration for universal automatists.

1.2: A New Kind of Science

By way of giving the notion of computation a little more texture, I'll mention

two slightly counterintuitive facts. And then I'll describe Stephen Wolfram's four classes of computation.

The first counterintuitive fact is that just because an output is computable doesn't mean it's easy to arrive at. Computer scientists use the word *feasible* in this connection.

- *Informal Definition.* A particular computational process is *feasible* if it produces the desired result in a humanly reasonable amount of time.

A computation that you can do by hand in a few minutes is feasible, something that would take you ten years isn't feasible. In other words, a computation is unfeasible if carrying it out would take an unreasonable amount of resources and/or an unreasonable amount of time.

- *Counterintuitive fact.* Although a computation may be theoretically possible to carry out, it can be practically unfeasible to do so.

This is rather obvious, but it's worth remembering. Sometimes we get carried away by a proof that one system can in principle simulate some other system, and we lose sight of the fact that the simulation is in fact so slow and cumbersome that it's quite unfeasible. Most artificial intelligence (AI) programs fall into this category vis-à-vis the human mind—yes, they can simulate some small parts of human reasoning, but the simulations are so slow that applying them to realistically large inputs is unfeasible. (Actually, the situation is worse than that; not only are our existing AI programs unfeasible for large problems, we probably haven't found the right *kinds* of AI programs at all.)

The feasibility of a computation depends both on the computational system you plan to use and on the computational method you plan to employ. This relates to the distinction between hardware and software. If you have very slow and clunky hardware, almost no computations are feasible. But no matter what your hardware is, improved software (such as clever calculating tricks) may expand your arena of feasibility.

Suppose we agree with the universal automatists that most physical processes are computations. By and large these physical computations are unfeasible for our personal computers. Not only is it unfeasible to digitally

emulate the global weather; even simulating the turbulent flow of tap water is beyond our existing electronic machines. But—and this is my point—the processes may well be computations anyway.

For people, the most admired and intimately familiar computations of all are the creative and meaningful processes of human thought. For reasons I'll explain in CHAPTER FOUR: *Enjoying Your Mind,* I would not expect to see human creativity becoming a feasible electronic computation anytime in the next hundred years. But, again, this doesn't rule out the option of viewing the human brain as a type of computer just as it is. The brain is a system obeying a finite set of rules. Human thought is certainly a feasible computation *for the human brain,* it's just not currently feasible *for electronic computers*.

The second counterintuitive fact I want to mention is that computations can yield genuine surprise. One might suppose that a deterministic rule-based process must flow along in quite a routine fashion. Yes, but this doesn't mean that the long-term behavior of the computation is predictable.

- *Informal Definition*. P *is predictable* if there is a shortcut computation Q that computes the same results as *P,* but very much faster. Otherwise *P* is said to be *unpredictable*.

A more precise definition of what I mean by unpredictable can be found in the Technical Appendix at the end of the book. But the basic idea is that if *P* is unpredictable, there is no dramatically faster way to get *P*'s output other than to carry out the computation of *P*.

As a really simple example of a predictable computation, suppose you want to decide if an input number is even or odd. A slow way to compute this would be to painfully carry out a long division of two into the number, working out the whole quotient on the way to finding out if the remainder happens to be zero or one. A fast, shortcut way to compute the same information is just to look at the last digit of your input number, and say that the number is even if the last digit is zero, two, four, six, or eight. The slow long-division computation of evenness is predictable in our sense because the much faster last-digit computation produces the same results.

In practice, anyone who writes computer programs for a living is going to try to make the code as efficient as possible. This means that, in practice, most of

the PC programs we work with will in fact be unpredictable, in that there's no way to drastically speed them up. As it happens, unpredictability is relatively common across the whole spectrum of possible kinds of computation.

- *Counterintuitive fact.* Many simply defined computations are *unpredictable*.

If I call this counterintuitive, it's because, before you analyze the notion, you'd expect that computations *would* be predictable, at least in the colloquial sense of being dull and unsurprising. Indeed, "Don't act like a robot," means something like, "Don't be so predictable." Given the deterministic nature of a rule-based computation, it's true that, step-by-step, the computation is predictable. Given *A* we always get *B,* given *B* we always get *C,* and so on.

But—and this is the point that takes some getting used to—there's often no shortcut method for looking ahead to predict the end result of a computation. That is, if I want to know the end result of a billion-step computation, it's very often the case that there's no faster method than carrying out the billion-step computation itself.

Consider an analogy. When Columbus sailed across the Atlantic, it was predetermined that he'd find the West Indies (which, to his dying day, he thought were part of Asia). But Columbus never could have predicted the shapes of those islands (whatever he called them) without making the trip.

The unpredictability of computations becomes noticeable when we throw substantial problems at our machines. A famous example involves *pi,* the numerical value of the ratio that a mathematical circle has to its diameter. This number, which begins 3.14159 . . . , is known to have a decimal expansion that goes on and on without ever settling into a pattern. Nevertheless, there are simply defined computational methods for finding the successive digits of pi by means of multiplications, additions, and the like. One (not very efficient) approach would be to sum together more and more terms of the endless alternating series:

$$4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - \ldots$$

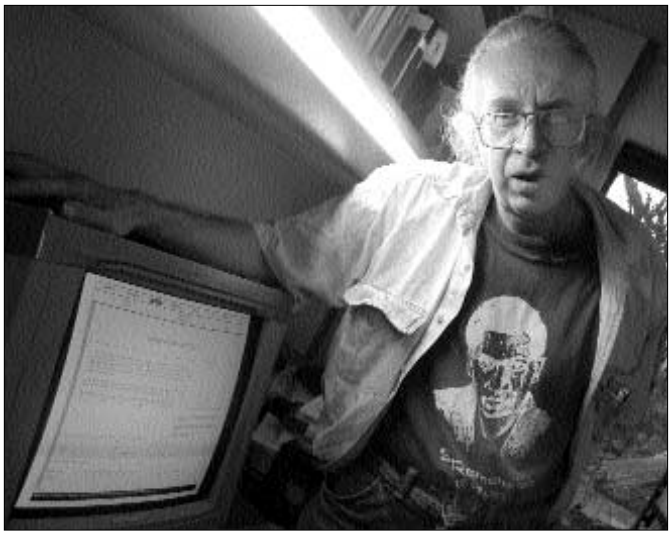


Figure 6: William Gosper, Programmer King

The face on Gosper's T-shirt is that of the prolific Hungarian mathematician Paul Erdös

In the mid-1980s, my old-time computer fanatic friend William Gosper once held a world record for computing pi. He calculated it out past the seventeen millionth digit. I've copied here from one of Gosper's e-mails the first hundred digits after the seventeen millionth place: 6978965266 4312708718 8987022892 7339840950 1815706767 7105940124 6541910101 0611655655 1475202499 7781719847.

Conversations and e-mail from Gosper have been touchstone experiences for me ever since moving to Silicon Valley. He's like the last exemplar of some extinct species of bird, a chatty apteryx in his aboriginal nest, surrounded by antique plastic artifacts, such as an ellipsoidal electric pencil sharpener, a stack of Symbolics computer monitors, and a mound of numbered Aerobie disks. I should mention that he feels the best way to truly compute pi is to express it as an enormous tower of nested fractions, which is what he actually did to net his particular catch of pi. It was only so as to be able to compare his work with the work of others that he reduced his tower of pi to base ten digits in a process that he calls, somewhat disdainfully, "decimalizing pi."

In any case, before Gosper's calculation was done, there was no way to know that, say, the seventeen millionth digit would be six. The only way to get Gosper's digits was to let a heavy-duty electronic computer munge on the problem for a long period of time. Yes, the value is predetermined by the laws of mathematics, but it's not really predictable.[4]

The notion of computer programs being unpredictable is surprising because we tend to suppose that being deterministic means being boring. Note also that since we don't feel ourselves to be boring, we imagine that we must be

nondeterministic and thus not at all like rule-based computational systems. But maybe we're wrong. Maybe we're deterministic but unpredictable.

I mentioned that unfeasibility is a relative notion, depending on the system you intend to use for running a given computation. Something's unfeasible on a given system if takes longer than you can reasonably wait. Unpredictability, on the other hand, is a more absolute notion. A computation is unpredictable if there is no other computation that does the same things a lot faster.

It's often enlightening to examine the possible interactions of newly defined properties. How do feasibility and predictability relate to each other if we temporarily limit our attention to computations on personal computers? As it turns out, all four possible combinations are possible.

- *Feasible and predictable*. These are the very simplest kinds of computation. I'm thinking here of a trivial computation like, say, multiplying seven by one thousand. Without getting out your pencil and paper, you know that $7 \times 1{,}000$ is 7,000. The computation is predictable. You know how it will turn out without having to carry out the details. But if you had to work out the details, you could, as the computation is feasible as well as being predictable.
- *Feasible and unpredictable*. These are the computations that interest computer scientists the most. Here there's a computation you can actually carry out, but there's no quick way to guess the result in advance. In a case like this, your computing system is doing something worthwhile for you. The computation is discovering a fact that you wouldn't have been able to guess.
- *Unfeasible and predictable*. Suppose that the computation was some very trivial task like replacing every symbol of an input string by zero. For any given input string, the output string is predictable: it will be a row of zeros the same length as the input string. But if the input string is going to be of some insane length—imagine a galaxy-spanning message a gazillion characters long—then there's no feasible way to feed

this thing into my desktop PC and expect an answer in any reasonable length of time. So in that sense the computation is unfeasible, even though the eventual output is predictable.

- *Unfeasible and unpredictable.* Rest assured that whatever is going on inside your head is both unpredictable and, relative to existing electronic computers, unfeasible. But you're doing it anyway. As Frank Zappa used to say, "Ain't this boogie a mess?"

One of the main themes in *The Lifebox, the Seashell, and the Soul* will be that computations come in certain basic flavors. This is the whole reason why it might be worthwhile to think of things like flowers, thunderstorms, and orgasms as computations. Yes, the details of these computations must elude us, and any simulation of them would be unfeasible. Even so, there are certain properties such as unpredictability that can be usefully applied to real-world phenomena.

We're going to be saying a lot about a very useful classification of computations that was invented by Stephen Wolfram in the 1980s. Wolfram noticed that there are four main behaviors for arbitrary computations that are left running for a period of time.

- *Class one*. Enter a constant state.
- *Class two*. Generate a repetitive or nested pattern.
- *Class three*. Produce messy, random-looking crud.
- *Class four*. Produce gnarly, interacting, nonrepeating patterns.

It's pretty easy to understand what class one and class two computations look like, although it's worth mentioning that a regularly branching pattern would also fall under class two. The essence of being a class two computation is that the outputs don't generate surprise.

My hacker friend Gosper refers to class three style patterns as "seething dog barf." These are messy, random-looking computations with no obvious order or structure in their outputs.

Class four computations, on the other hand, generate images more like never-repeating lace. Class four computations might be characterized as having behavior that appears purposeful. I like to use the word *gnarly* for

class four processes—gnarly in the sense of twisting tree roots, large ocean waves, or weathered human faces.

When a rough distinction will do, we speak of class one and class two computations as being *simple* and class three and class four computations as *complex*. Figure 7 summarizes the terminology.

The borders between the computation classes aren't very crisp. There are times when the distinction between class three and class four isn't obvious. And it's not always clear if a system is class two or class four—consider the fact that some systems can appear interesting for a very long time and only then settle down to being periodic. We don't always know whether we can in fact find a "good" input that will keep a certain computation running and producing novelty forever, thus showing that it really is a class four rule.

Wolfram's critics complain that his computation classes aren't formally defined and that, when we do attempt formal definitions, determining the class of a computation can be an unsolvable problem ("unsolvable" in a certain formal sense that I'll describe in chapter 6). Wolfram might reply that using rough-and-ready concepts is typical for a newly developing branch of science. I agree with him. I think his critics miss the forest for the trees. With an open mind you can indeed distinguish the four computation classes; you'll begin to see this as our examples accumulate. Yes, there will be some borderline cases, but that doesn't mean the classes don't exist.

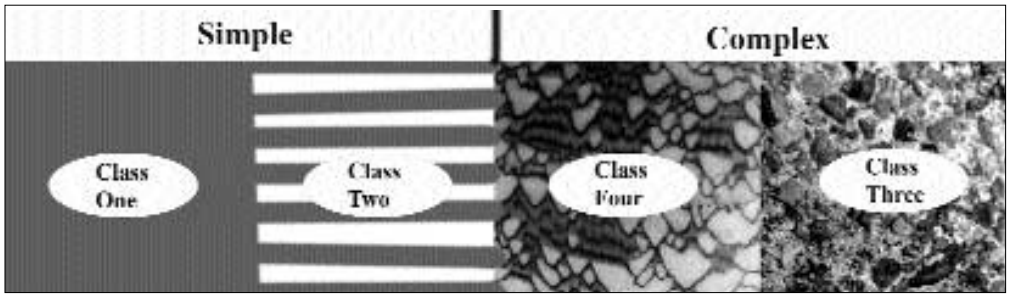


Figure 7: The Spectrum of Complexity

A simple computation is in class one or class two. A complex computation is in class three or class four. Despite the number order of the names, the gnarly class four in some sense lies in between the periodic class two and the random-looking class three.

Wolfram's initial investigations all had to do with feasible computations—in that he was looking at actual programs he could run. But his classification system applies equally well to the enormous computations carried out by physics and biology. The limitations of our digital silicon machines are such that we can't feasibly emulate any really large parts of the real world. Even so, it's very useful to categorize the unfeasible-for-the-PC computations that we see all around us.

Some natural phenomena die out or become static—these are the computations of class one. Other aspects of the world are periodic or class two—one thinks immediately of the rising and setting of the sun or the ebb and flow of the seasons. The class three aspects of the world are the seemingly random ones—you might think of radio hiss or TV-screen snow. But the most interesting computations in nature are all class four. As we'll see, examples include the forms of clouds and of trees, the flow of your thoughts, and the spacing of cities upon a map.

Wolfram has made two conjectures about his computation classes. The first is the Principle of Computational Equivalence (PCE for short).[5]

- *Principle of Computational Equivalence (PCE).* Almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication.

What he means by this is that, in a sense that we'll make precise later on, all of the class three and class four computations are equally complex. Rather than believing that some complex computations are simpler than others, Wolfram feels that nearly all of them are of an equal and maximal complexity.

The PCE is in some sense discouraging, as it seems to tell us that when you can't see a simple explanation for a natural phenomenon, this means that the phenomenon is not only complex, but of a maximal complexity. Anything that's not obviously simple is in fact very gnarly.

A quick example. Consider the motion of the leaves on a tree. A physicist might describe the system as a wind-driven multiple-pendulum system. But the resulting computation is class four and certainly complex. If the PCE holds, then the gnarly motions of the leaves are to be as sophisticated as what's going on inside my brain. I seem to be a fluttering leaf? Maybe so.

Besides his PCE, or Principle of Computational Equivalence, Wolfram advocates a second conjecture, which I call the PCU or Principle of Computational Unpredictability.

- *Principle of Computational Unpredictability (PCU).* Most naturally occurring complex computations are unpredictable.

Here again, complex means class three or class four. And, as I mentioned before, when I say that a computation is *unpredictable,* this means there's no drastically faster shortcut computation that will reliably predict the given computation's outputs.

When we find some kind of natural process going on, we can often model the process as a computation. And in certain rare cases, we can also model the process by some simple and rather easily solvable equations. The PCU says that the latter situation is exceedingly rare. Generally speaking, there is no quick way to predict the results of a naturally arising computation.

Wolfram doesn't feel a need to explicitly state the PCU, but it's implicit in *A New Kind of Science*. He prefers to use the words *reducible* and *irreducible* for what I'm calling *predictable* and *unpredictable*—I insert some bracketed phrases in the following quote to keep this clear.[6]

> So what this [The Principle of Computational Equivalence] means is that systems one uses to make predictions cannot be expected to do computations that are more sophisticated than the computations that occur in all sorts of systems whose behavior we might try to predict. And from this it follows that for many systems no systematic prediction can be done, so that there is no general way to shortcut their process of evolution, and as a result their behavior must be considered computationally irreducible [or unpredictable].
>
> If the behavior of a system is obviously simple—and is say either repetitive or nested—then it will always be computationally reducible [or predictable]. But it follows from the Principle of Computational Equivalence that in practically all other cases it will be computationally irreducible [or unpredictable].

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

As I'll discuss in CHAPTER SIX: *Reality Upgrade:* the PCE and the PCU are in fact independent of each other. While the latter is used to deduce the unpredictability of naturally occurring processes, the former is used to deduce the *unsolvability* of certain questions about these processes—where unsolvability means that certain kinds of questions can't be solved by any conceivable kinds of computation at all.

I agree with Wolfram that both the PCE and the PCU are likely to be true for all of the interesting examples of naturally occurring computations—including physical systems, biological growth, the human mind, and the workings of human society.

Before closing this section, I want to introduce one more concept. When a computation generates an interesting and unexpected pattern or behavior, this is called *emergence*. I'll give three quick examples drawn from, respectively,

Figure 8: Simulated Bird Flocking

the fields known as artificial life, fractals, and cellular automata. (Most of the computer graphics figures in this book were made with programs I had a hand in authoring; see the Image Credits section at the end of the book for details.)

In artificial life, computers try to simulate the behaviors of living organisms. A classic discovery in this field is the *boids* algorithm by Craig Reynolds. Reynolds found that if a group of simulated birds, or boids, obeys a few simple rules, the boids will seem to move about as a coherent flock. This is an example of emergence in that the somewhat unexpected flocking behavior emerges from the collective computations carried out by the individual boids as suggested in figure 8. I'll say more about the boids algorithm in CHAPTER FIVE: *The Human Hive.*

A fractal is a structure that has interesting details at many different levels. The most famous fractal is the *Mandelbrot set* (figure 9). Suppose that we think of a computer screen as a region of the plane, with each pixel representing a pair of real numbers. Suppose further that for each pixel we use the corresponding number pair as an input for an iterated computation that terminates by specifying a color for the pixel. In the 1970s, Benoit Mandelbrot investigated a wide class of such computations that produce wonderfully intricate fractal patterns. Being a fractal, the Mandelbrot set has the property that one can zoom in on it, discovering level after level of detail. This is an example of emergence in that we have a cornucopia of forms arising from iterated applications of a very simple rule.

I'm going to say a lot about cellular automata in this book; they're a fascinating type of computation popularized by Stephen Wolfram. For now, think of a two-dimensional cellular automaton as a computation in which each pixel on a computer screen simultaneously updates its color according to the same rule. What gives the process its punch is that each pixel is allowed to look at its neighbor pixels. As a simple example of a such a cellular automaton rule, suppose that each pixel is black or white, and that a pixel updates itself by polling its nearest neighbors as to whether the majority of them are white (figure 10). It turns out that if you use an algorithm of awarding close elections to the losing side, a random sea of black-and-white pixels congeals into smoothly undulating globs, not unlike the droplets in a lava lamp. The high-level globs emerge from

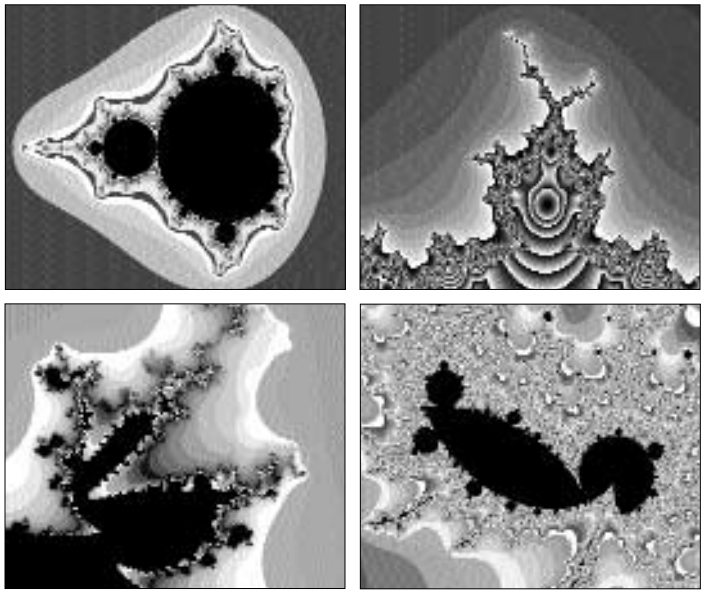


Figure 9: Mandelbrot Sets

Left to right and top to bottom, we have the traditional Mandelbrot set based on a formula of the form $z = z^2 + c$; *a zoomed-in view of the upper topknot of this set with an additional algorithm used to fill in the black region; a detail of a cubic Mandelbrot set based on a formula of the form* $z = z^3 + bz + c$; *and a detail of the so-called Rudy set, which is based on the family of cubic Mandelbrot sets. To me the last three images resemble, respectively, Ronald Wilson Reagan dressed as Bozo the Clown, a roaring dragon, and a friendly little rocking-horse.*

the low-level interactions of the cells. We call this the Vichniac Vote rule or just the Vote rule.[7]

The essence of the flocking, the Mandelbrot and the Vote computations is that something interesting emerges from a simple rule and a generic starting condition.

Emergence is different from unpredictability. One the one hand, we can have unpredictable computations that don't have any high-level emergent patterns: the dull digits of pi would be an example of this. On the other hand, we can have computations that generate emergent patterns that are, in the long run, predictable.

If you let the Vote rule run long enough, one color or the other melts away,

Figure 10: The Vichniac Vote Rule

Each pixel is treated as the center of a 3 × 3 grid of nine cells. The new color of a pixel is white if the total number of white pixels in its neighborhood grid is four, six, seven, eight, or nine, and the new color of the pixel is black if the total number of white pixels is zero, one, two, three, or five. The three images show a random initial start with 320 × 200 pixels, the appearance of the system after thirty updates, and the appearance after three hundred updates.

leaving a blank wasteland with perhaps a few tiny, rhythmic blinkers. The Vote rule is ultimately a predictable class two computation.

How about flocking and the Mandelbrot set? In most situations, the flocking behavior of a group of simulated birds will be class four and unpredictable, with new flocking configurations emerging from time to time—one such pattern I've observed is that sometimes a pair of birds will circle each other in a pattern like a tightly coiled double helix. And if you were allowed to endlessly zoom in on the emergent fractals of the Mandelbrot set, I think you'd also find unpredictable class four behavior, at least in the regions near the boundary of the set (although whether the Mandelbrot set is truly unpredictable is, I believe, an open problem).

1.3: Reckoning a Sum

Human calculation is the original model for the notion of computation, so it's well worth analyzing how we use a pencil and paper to calculate something like 275 + 484. Before reading ahead, you might carry out the sum yourself, paying attention to what goes through your mind.

$$\begin{array}{r} 275 \\ +\ 484 \\ \hline \end{array}$$

My own thoughts go something like this:

"This is an addition problem, so I'll use the adding routine I learned in grade school.

"I'll start at the top of the right-most column and work down, reading numbers and keeping a running sum in my head.

"That first mark is a five and the next one down is a four. The five looks like a fat man wearing a billed cap. Maybe he's a train engineer.

"Five plus four is nine. How do I know that? Because I memorized the simple sums fifty years ago in Kentucky. My God, where has the time gone? Our teacher was Mrs. Graves, the choirmaster's wife. There was that one boy, Lee Tingley. He couldn't learn the sums and he'd always sneak and count on his fingers. I did that, too, sometimes, pretending just to be drumming my fingers on the table—but Mrs. Graves had a sharp eye for finger-counting. She was strict. Kind of a pioneer type. She and her husband lived in a log cabin with their four kids. How does adding on your fingers work? Well, it depends on knowing the order of the counting numbers—if you don't know that you're lost. Anyway, let's see, I'm in the middle of an addition, and five plus four is nine.

"There's no more numbers in the column, so I write nine at the bottom and shift my gaze to the top of the next column to the left.

"Seven plus eight is fifteen. Write five and remember to carry one. I've always been a little uneasy about carrying, and borrowing is worse, especially borrowing from zero. I never fully understood borrowing until graduate school. Better not to think about it too much, just use the rules Mrs. Graves drummed into me. I'm carrying a one.

"I'm looking at the top of the next column to the left. I see a two. The carried one plus the two makes three. Remember that. The next number down is four. My three plus the four makes seven. Write seven.

"There's no more columns, so I'm done. 759 means seven hundred and fifty-nine. That's the answer."

If you do a lot of arithmetic by hand—not that many of us do anymore—then all of this is quite automatic. Indeed, arithmetic seems hard exactly when you're so rusty at it that you have to consciously think about what you're doing.

Rather than speaking of a person doing pencil and paper arithmetic as a "computer" or "calculator," let's use the old-fashioned "*reckoner.*"

The reckoner's computation involves several levels of rules. Working our way down from the highest level, we start with the implicit behavioral rule that a reckoner looks at a piece of paper, decides on an algorithm, and then carries out the calculation. Not just any person would know to do this. Becoming a reckoner involves learning certain rules of behavior. These rules make up the "operating system" for pencil-and-paper arithmetic.

A level below that is the specific algorithm the reckoner uses, for instance, the standard procedure for adding numbers.

Deeper down are the memorized sum tables that the reckoner draws upon.

And even more basic is the reckoner's ability to read and write numbers.

We might also wonder about the underlying biology that keeps the reckoner alive, about the physics that allows a pencil to make a mark on a piece of paper, and about the background laws of logic that make all of this hang together in an orderly fashion.

Usually we like to think somewhat abstractly and take most of these elements for granted. But there does seem to be a sense in which a sizable little corner of the world gets dragged into something as simple as a child adding two numbers on a blackboard.

What we're seeing here is something I mentioned before: Real-world computations have many levels of rules.

It's instructive to view familiar things with a fresh sense of wonder. Consider a boy adding 275 to 484 to get 759. Look at him through alien eyes. The brown-eyed juvenile grasps a stick of diatomaceous matter with one of his clusters of articulated tentacles—ah yes, he's "holding chalk in his hand." He studies two groups of squiggles and scratches fresh squiggles below them. What does this mean? He's making a prediction about a certain possible counting behavior. He and his race have a rote routine for producing a distinct name for each ordinal number. "One, two, three, . . . two hundred and seventy-five, . . . four hundred and eighty-four, . . . seven hundred and fifty-nine." The boy's calculation demonstrates that if he were to count to 275, and then count 484 steps further, he would attain the number 759. His squiggle manipulations have compressed the work of counting through 759 numbers to less than a dozen elementary operations. Clever lad.

This brings out two key points about computations.

First of all, some computations are *equivalent* to each other in terms of what

they compute. For instance, I can add two numbers either by using arithmetic or by using an expanded version of "counting on my fingers." I get the same answer in either case, so the two computational methods are equivalent.

The second point is that equivalent computations can differ in how much time they take to carry out. If two different algorithms are used on one and the same system, it may be that one is always faster. Pencil-and-paper arithmetic is faster than counting.

The speed improvement you get by using faster software is independent of the gain you get by switching to a faster computer. Certainly a stage-performing calculating prodigy will be able to add numbers faster than our boy at the blackboard. But the prodigy, too, will add faster when using arithmetic than when working with brute-force counting.

How much time does arithmetic save? Allow me a brief geek-out on this topic. Using arithmetic instead of simple counting is an example of one computation being what computer scientists call "exponentially faster" than another. If a fast computation takes L steps and a slow computation takes on the order of 10^L steps, we say the fast one is exponentially faster than the slow one.

The relevance of this for our two styles of doing arithmetic has to do with the fact that, if an integer takes L digits to write, then the integer it represents has a size on the order of 10^L. Using digits can be exponentially faster than counting by ones.

As an example of a exponential speedup, suppose I wanted to reckon, let us say, the sum 123,456,789 + 987,654,321. This would be a matter of adding two nine-digit numbers.

$$\begin{array}{r} 123{,}456{,}789 \\ +\ 987{,}654{,}321 \\ \hline \end{array}$$

The pencil-and-paper reckoning of this involves summing nine columns. Adding each column will have some fixed cost of maybe ten primitive steps: three shifts of focus (from top digit, to bottom digit, to write slot, to top of the next column), two reads, two sum lookups (including adding the carry), a write operation, possibly carrying a digit to the next column, and a check to see if you're done.

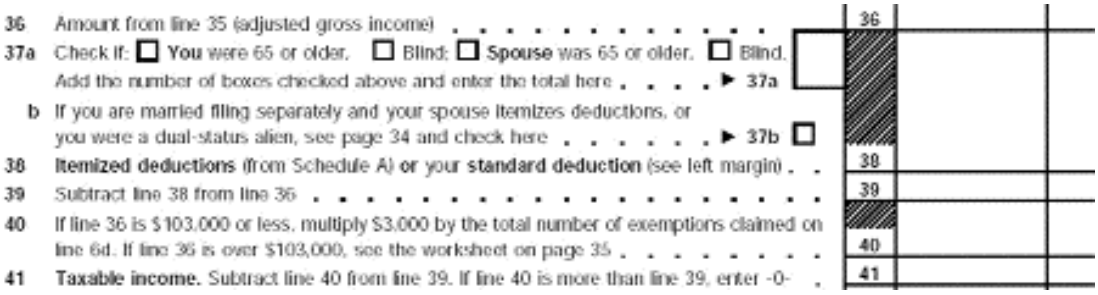


Figure 11: Excerpt from the U.S. Income Tax Form 1040

So using pencil-and-paper arithmetic to add a pair of nine-digit numbers requires no more than nine times ten steps, in other words ninety steps. That's a lot faster than counting by 123,456,789 by ones from 987,654,321 to arrive at 1,111,111,110—which would take over a hundred million steps, and isn't really feasible.[8]

By the way, this example also illustrates the point that something that is unfeasible for one style of computation may be feasible for a different kind of computation, even on one and the same system. Another point is that this particular addition problem has a very simple-looking answer, and that, with a little insight, a reckoner could have anticipated that and sped up the computation a bit more. But insight is an exceedingly difficult thing to automate.

By chaining together arithmetic problems a reckoner can carry out a very broad range of computations. What do I mean by chaining problems together? Consider a relatively complicated activity for which adults regularly use arithmetic: filling out tax forms. A tax form often embodies a linked chain of arithmetic problems.

Thus, you might be asked to write your income in row 35, write your deductions in row 38, write row 36 minus row 38 in row 39, write row 6d times 3,000 in row 40, write row 39 minus row 40 in row 41, and so on.

A list of instructions like this is a primitive example of what computer scientists call a *program*. Flexible beings that we are, we're able to handle a calculation task that contains not only numerical data to manipulate, but also instructions about the flow of the task.

It turns out that, given sufficiently elaborate instructions, we could carry out chains of arithmetic problems to compute the same results as a supercomputer.

Given enough time and patience, a human reckoner could carry out, say, all the ray-tracing and shading calculations needed to generate the frames of a feature-length computer-animated cartoon.

But of course the reality is that no reckoner *is* given that much time and patience. In order to make the best use of the computational worldview, we need to keep an eye on the distinction between abstract theoretical possibility and practical feasibility.

Now let's see how our electronic number-crunching machines expand our limits of feasibility. If nothing else, they allow us to be stupid faster.

1.4: Analytical Engines

If you use a standard file-exploring tool to poke around in the directories on your home computer, you find that certain areas of your hard drive contain data, such as images and documents, while other areas contain code for the software programs your machine runs. The high-level software is stored in one area (such as a Programs directory), the don't-touch-me-or-else low-level software in another (such as a Windows directory), and your documents are found somewhere like in a My Documents directory. The key fact is that both the software and the data are patterns of bits that are laid down in the memory. This is the *stored program architecture*. I mentioned that a tax form is a kind of program for a human reckoner. To say we are using a *stored* program architecture just means that we place a copy of the program into our machine's memory.

Why "architecture"? It's not like we're building the Parthenon here. Perhaps computer scientists use such a solid-sounding word to make up for the here-today-gone-tomorrow nature of their work. One of the less pleasant aspects of teaching computer science is how rapidly things change. Imagine if you were, say, a history professor, and, on showing up to begin your fall classes, you learn that this year your classes will be taught in Urdu, that instead of using markers on whiteboards you'll be using spray paint on rolls of butcher paper, and that your students will now be standing in the courtyard looking in through windows instead of sitting in your classroom. That's life as a computer science professor. No wonder we like to dignify our fly-by-night raree show with a moniker like "architecture."

Credit for the stored program architecture often goes to the Hungarian

émigré John von Neumann, who did much to promote the creation of the first digital computers in the late 1940s. In fact, this way of laying out computers is sometimes even called the *von Neumann architecture*. But von Neumann had a number of important collaborators, the idea of a stored program was already familiar to Alan Turing in the 1930s, and there are in fact foreshadowings of the stored program architecture as early as 1800, when people began to have the idea of changing a machine's behavior without having to mechanically rebuild it.

The Jacquard loom, invented by the Frenchman Joseph-Marie Jacquard in 1801, is programmable by punch cards. By coding up a tapestry pattern as a series of cards, a Jacquard loom is able to weave the same design over and over, without the trouble of a person having to read the pattern and set the threads on the loom.

In the mid-1800s a colorful Briton named Charles Babbage hit upon the idea of using punch cards to control computations. Babbage actually owned a woven silk portrait of Jacquard that was generated by a loom using 24,000 punch cards. (see figure 12.)

Babbage began by designing—but never quite completing (to read Babbage's memoirs is to want to choke him very hard)—a gear-based device known as a Difference Engine, which was to be used for calculating and printing out mathematical tables of logarithms and trigonometric functions, astronomical tables giving the computed positions of celestial bodies at various times, and life-insurance tables giving the expected earnings or annuities of people of various ages. In each case it was a matter of applying a particular algebraic formula over and over.

Figure 12: Drawing of the Woven Portrait of Joseph-Marie Jacquard

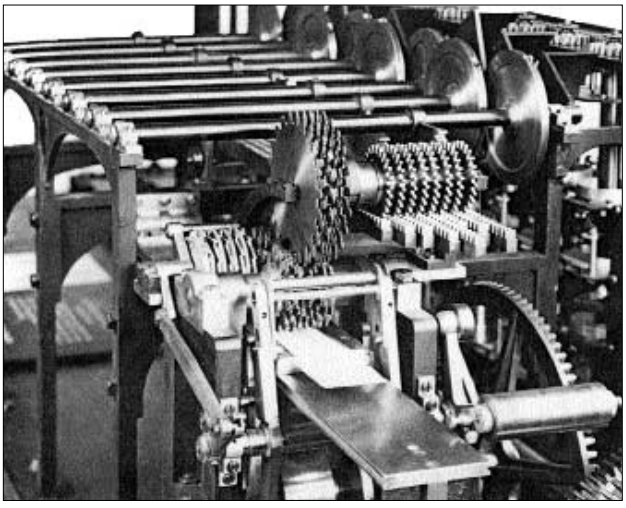


Figure 13:
A Detail of Scheutz's Difference Engine

There was a small but real market for a Difference Engine and eventually the Swedish inventor Georg Scheutz did actually complete and market two working Difference Engines (figure13). Rather than being envious, the big-hearted Babbage encouraged Scheutz and helped him sell his first machine to an astronomical observatory in Albany, New York.

One reason that Babbage never finished his own Difference Engine was that he was distracted by dreams of an even more fabulous piece of vaporware, a machine he called the *Analytical Engine*.

Babbage's description of the Analytical Engine may well be the very first outline for a calculating device where the program is separate from the action of the machinery. The Analytical Engine was to have a "mill" (think "chip") that executed arithmetic operations, and was also to have a "store" that would provide a kind of scratch paper: short-term memory for temporary variables used by the calculation. Babbage's then-novel idea was that the actions of the mill were to be controlled by a user-supplied program that was coded into punch cards like the ones used by the Jacquard loom. If we think of the deck of punch cards as being a kind of machine memory, Babbage's design foreshadows the stored program architecture—but it's not quite there yet.

One of the most lucid advocates of Babbage's Analytical Engine was the young Ada Byron, daughter of the famed poet. As Lady Ada memorably put it,

> The distinctive characteristic of the Analytical Engine, and that which has rendered it possible to endow mechanism with such extensive faculties as bid fair to make this engine the executive right-hand of abstract algebra, is the introduction into it of the principle

> which Jacquard devised for regulating, by means of punched cards, the most complicated patterns in the fabrication of brocaded stuffs. . . . We may say most aptly, that the Analytical Engine *weaves algebraical patterns* just as the Jacquard loom weaves flowers and leaves.[9]

In reality, Babbage's Analytical Engines were never built. But it's interesting to think about such engines—it brings home the idea that computers don't have to be boxes of wires and chips. Remember, a computation is any system with a process that is governed by a finitely describable set of rules.

In 1991, my fellow cyberpunk science-fiction writers William Gibson and Bruce Sterling published a fascinating alternate history novel, *The Difference Engine,* which imagines what Victorian England might have been like if Babbage had been successful. (Despite the title, the book is really about Analytical Engines rather than Difference Engines.) Just as our present-day computers are run by hackers ("hacker" in the sense of "fanatical and resourceful programmer," as opposed to "computer criminal"), the Analytical Engines of Gibson and Sterling are tended by "clackers." Here's their description of a visit to the Central Statistics Bureau in their what-if London:

> Behind the glass loomed a vast hall of towering Engines—so many that at first Mallory thought the walls must surely be lined with mirrors, like a fancy ballroom. It was like some carnival deception, meant to trick the eye—the giant identical Engines, clock-like constructions of intricately interlocking brass, big as rail-cars set on end, each on its foot-thick padded blocks. The whitewashed ceiling, thirty feet overhead, was alive with spinning pulley-belts, the lesser gears drawing power from tremendous spoked flywheels on socketed iron columns. White-coated clackers, dwarfed by their machines, paced the spotless aisles. Their hair was swaddled in wrinkled white berets, their mouths and noses hidden behind squares of white gauze.[10]

In the world of *The Difference Engine,* one can feed in a punch card coded with someone's description, and the Central Statistics Bureau Engines will spit out a "collection of stippleprinted Engine-portraits" of likely suspects.

Babbage's ideas bore fruit after a century. It was 1945 when von Neumann began promoting the stored program architecture, after working with the designers of a machine called ENIAC at the Moore School of Engineering of the University of Pennsylvania. Although it wasn't made of gears, the ENIAC was really a Babbage-style Analytical Engine. The ENIAC is sometimes regarded as the first general-purpose electronic computer, but it wasn't quite all the way there, in that its program wasn't stored in electronic memory. The ENIAC program was on a deck of punch cards; the machine needed to consult them every time it needed a program instruction.

A parenthetical note. Although ENIAC was originally meant to compute artillery trajectories, World War II was over before it started working. One of the first big computations ENIAC carried out was in fact a Cold War calculation to test the feasibility of building a hydrogen bomb: a numerical solution of a complicated differential equation having to do with nuclear fusion. It is said that the calculation used an initial condition of one million punch cards, with each punch card representing a single "mass point." The cards were run through ENIAC, a million new cards were generated, and the million new cards served as input for a new cycle of computation. (My guess is that the cards represented points arranged in a cubic grid a hundred units on a side, and that their values were updated on the basis of their neighbors' values.) You might say that the very first electronic computer program was a simulation of an H-bomb explosion. What a shame. Better they should have been looking at fractals, or simulating a human heart!

Programming the ENIAC involved making a deck of punch cards, manually arranging the wires on a plugboard, and setting a bunch of ten-position dials. There had to be a better way. As Arthur Burks, Herman Goldstine, and John von Neumann wrote in, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument,"

> Conceptually we have discussed . . . two different forms of memory: storage of numbers and storage of orders. If, however, the orders to the machine are reduced to a numerical code and if the machine can in some fashion distinguish a number from an order, the memory organ can be used to store both numbers and orders.[11]

The stored program architecture means that, in a certain sense, the high-level software is a kind of data that's processed by the low-level software that controls the host machine's basic functioning.

It's thanks in part to the stored program architecture that each of today's computers is in some sense equivalent to any other. If you have a Macintosh, you can get a Windows emulator that will allow your machine to read and execute Windows programs. If you're nostalgic for the PDP-1 computer used by the earliest computer hackers at MIT, you can search the Web and find a Java program that, when loaded on your machine, will allow it to emulate a PDP-1.

I've always loved that word, *emulate.* As humans we often try to emulate our heroes, that is, to learn a set of behaviors that make us be "just like" the hero. In effect, we're loading software into our brains. After watching a movie with a character I find particularly interesting, I'll often spend a few minutes emulating this character—seeing through the character's eyes, moving as the character moves, thinking as the character seemed to think. Books and other works of art have this effect, too, but there's something especially hypnotic about films.

Emulation generalizes the stored program concept. To be precise, we say that a computation Big emulates another computation Small if you have a special auxiliary input emulatesmall so that the states produced by Small(In) are the same as the states produced by Big(emulatesmall, In). In this situation we speak of emulatesmall as an emulation code.

Before making this more precise, let's recall how we're thinking of computations.

We view a computation *P* as a process that we set into motion by giving it an input In. Thus *P*(In) is a process that changes as time *t* increases. To be quite general, we're allowing both for the possibility that *t* increases in abrupt steps, as in a digital computer, and for the possibility that *t* is continuous, as in a physical system like a fluttering leaf. We write *P*(In, *t*) = Out to mean that after time *t*, the computation *P*(In) is in state Out. And we assume that we have some method Is*P*Done(Out), called a target detector, that allows us to decide if the computation is to be viewed as having halted when it reaches the state Out. Let's adopt the following additional terminology.

- *P*(In) *produces* Out means that for some *t, P*(In, *t*) = Out.
- *P*(In) *returns* Out means that for some *t, P*(In, *t*) = Out and Is*P*Done(Out) is True.

And now we can define emulation.

- *Definition of Emulation.* Big *emulates* Small if there is an emulation code emulatesmall such that for any states In and Out,
  Small(In) returns Out if and only if
  Big(emulatesmall, In) returns Out.

So Big emulates Small means that having access to Big and the emulation code emulatesmall is as good as having access to Small.

The definition of emulation is rather technical, but the concept is a natural one. Let me suggest some analogies.

- Think of Big as a PC and Small as a pocket calculator. Big comes equipped with a calculator accessory that acts as an emulatesmall to make it behave just like a calculator.
- Think of yourself as Big and me as Small. The book you hold is meant to serve as an emulatesmall that allows you to emulate my thoughts.
- Think of Mr. Big as a man, Ms. Small as a woman, and emulatesmall as a dress. If Mr. Big wears a dress, can he reproduce all the behaviors of Ms. Small? No. Mr. Big will never give birth to a baby. So he can't presently be said to emulate Ms. Small. But hold on. Maybe at some future time, men may gain the ability to grow cloned offspring of their own. And in this event, perhaps Mr. Big *can* be said to fully emulate Ms. Small.
- Think of Big as a tree branch rocking in the wind and Small as a PC. I'm of the opinion that the Big branch's behavior is rich enough to emulate anything that the Small PC can do. In order to make the definition of emulation apply, however, I'd need to incorporate some method of translating from the binary language of machines into the positional "language" of leaf and branch positions. I take up the issue of translations in emulations in the Technical Appendix.

I mentioned above that any one of our personal computers can emulate any other. This is perhaps a bit surprising. After all, if you were to believe some of the ads you see, you might imagine that the latest PCs have access to new, improved methods that lie wholly beyond the abilities of older machines. Could there be a new machine with such tricky goodies on its chips that an older machine would not in fact be able to load up and execute emulation software for it?

Well, if the emulation program for the new machine is so large that it wouldn't be able to fit into my old machine's memory, then, no, the old machine can't emulate the new one. But this is a hardware limitation that seems peripheral to the core issue of functional capability. If I'm allowed to equip my old machine with as much additional memory I need, then yes, I can always get it to behave like any other computer at all.

This somewhat surprising fact has to with a phenomenon that computer scientists call universality. It turns out that many computations can in fact emulate any other computation. We call these maximally powerful computations *universal*.

- *Definition.* A computation is *universal* if it can emulate any other computation.

Now, you might expect it to be fairly hard to get a computation to be universal. But nothing could be further from the truth. Universality is easy. Once *any* computational system advances past a certain very low threshold, it becomes universal. How low is the threshold? Being able to add and multiply is more than enough. And, as we'll see, even more rudimentary capabilities will do.

In point of fact, when we examine the naturally occurring computational systems around us—like air currents, or growing plants, or even drying paint—there seems to be reason to believe that the vast majority of these systems support universal computation. This belief is part of the content of Wolfram's PCE: If some complex computations are universal, and most complex computations are of equivalent sophistication, then most complex computations are universal.

Universality is a big deal. The existence of universal computation means

that there is a maximal level of computational complexity. And the ubiquity of universality means that this maximum is rather readily attainable. Computation is in some sense already as good as its going to get. We're in a position a bit like someone who's inherited a fortune of a vastness they're still learning to understand.

1.5: The Tiniest Brains

Starting with thoughts about arithmetic, Alan Turing formulated a minimally simple definition of computation in the 1930s—well before any real electronic computers had been built. Turing's approach was to describe an idealized kind of computer called a Turing machine.[12]

In practice, nobody builds Turing machines. They're so primitive that even adding numbers can be unfeasibly time-consuming with one of these devices, and programming such a device to do anything complex is mind-numbingly dull.

Nevertheless, there are several good reasons for learning about Turing machines.

First of all, many Turing machines are universal, that is, they can, however slowly, carry out any possible computation. Looking at Turing machines helps us understand how little is really needed for universal computation.

Second, the design of a Turing machine resembles the design of an electronic computer, albeit in embryonic form. Understanding Turing machines is a good preparation for understanding PCs.

Third, the rudimentary quality of Turing machines makes them easy to think about. By searching through all possible Turing machines we can in some sense search through all possible computations. In his original paper on the topic, Turing proved that no Turing machine can distinguish between the true and false theorems of mathematics, which in turn showed that mathematical truth is in some sense undecidable for any computer at all. More recently, Stephen Wolfram has carried out a series of computer searches over the class of Turing machines to help confirm his hypothesis that computations come in only four flavors: they die out, they repeat, they seethe messily, or they create gnarly patterns.

So, all right, what's a Turing machine?

I once looked through a specification of the librarians' Dewey decimal

system and found there is actually a classification for "Turing machines, manufacture and distribution of." But in point of fact, Turing machines are not real physical devices that people build. They're idealized models of an extremely simple kind of digital computer. Turing's original inspiration for the Turing machine was to try to capture the behavior of a human reckoner—but without all the squishy stuff on the inside.

To begin with, a Turing machine has only some finite number of internal states. These are analogous to a reckoner's mental states, such as the state of remembering to carry a 1.

As a further simplification, a Turing machine uses a linear tape of cells instead of a two-dimensional grid of paper. A Turing machine focuses on one cell at a time on its tape; more concretely, we think of the machine as having a read-write head that moves from cell to cell.

During each update, the machine reads the symbol in the cell, possibly changes the symbol in the cell, moves its head one cell to the left or one cell to the right, and enters a new internal state. Having completed one update step, it begins the next: reading the new cell, changing it, moving its head, and altering its internal state once again.

What determines the Turing machine's behavior? We can look at it this way: each stimulus pair of (*internal state, read symbol*) leads to a unique response triple of (*write symbol, move direction, new state*). The high-level software for a Turing machine is a lookup table that supplies a response triple for each possible stimulus pair.

A Turing machine's input is a string of symbols on the tape. Suppose we simply write *d* to stand for a tape with a particular symbol pattern that we can also call *d*. We set a computation in motion by putting the machine into its starting state and setting its head on the leftmost nonblank symbol of *d*. An output is any resulting pattern of symbols that appears on the tape at a later times.

Figure 14 represents a Turing machine in action. It uses only two symbols, the white cell and the black cell, which we might also think of as zero and one, and it has three states. Each row of the figure shows a picture of the Turing machine's tape, with time running down the page from top to bottom—that is, the starting configuration is the top line and the later configurations are below. The picture also includes small representations of the

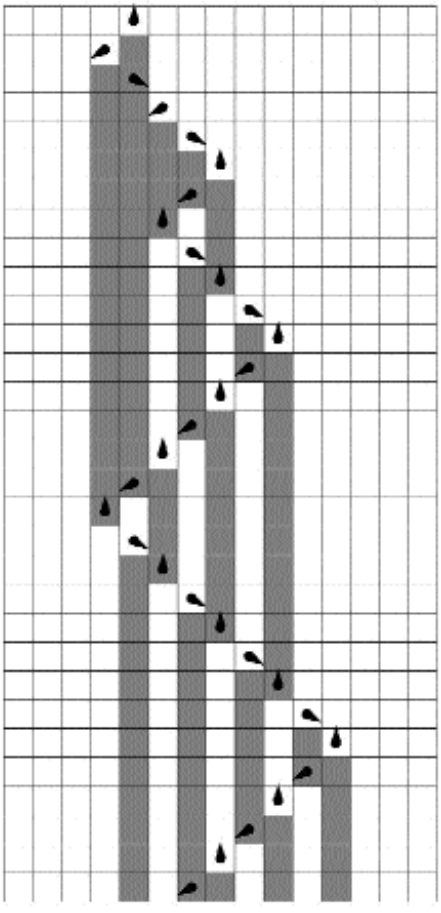


Figure 14:
A Turing Machine History

This is a three-state Turing machine.

Turing machine's head as a little pointer whose direction indicates the internal state that the Turing machine is in during that particular snapshot. (This useful method of representing Turing machines was introduced by Stephen Wolfram in *A New Kind of Science*.)

The particular machine depicted starts on a blank tape and endlessly shuttles back and forth, filling the tape with an ever-growing pattern of alternating marked and unmarked cells. It never stops. Pointless, you may say, but, hey, it's a computation!

In some applications of Turing machines we are concerned with finding cases where the machine halts, that is, reaches a state after which the output pattern doesn't change any further. Although its possible to do this by having the machine go into an endless loop without writing or erasing anything more, most discussions allow computations to have a special "halted" state, and specify that once a Turing machine enters its halted state, it stops looking up further moves.

Some discussions of Turing machines focus almost exclusively on machines whose computations halt. But in *The Lifebox, the Seashell, and the Soul,* we're equally interested in open-ended computations that are willing to run for as long as you let them. In Wolfram's terms, a computation that halts for every input is class one. Naturally occurring class four "computers," like the weather, the plants, or our minds, all have the quality of being willing to continue indefinitely. It's only a destructive external input that brings most natural computations to a halt—as when, for instance, a toxic spill eliminates a patch of plants, a cerebral hemorrhage cuts off a person's thoughts, or a sun explodes and puts an end to its planets' weather.

As I mentioned earlier, we also have a notion of halting for arbitrary computations *P*. Here we can have a *target detector,* Is*P*Done, that has two special output states, True and False. Is*P*Done is a helper computation that allows us to distinguish certain *target states* as being states in which P has produced an answer. So as to block off an endless regress, we require that there be no problems in trying to decide when Is*P*Done itself has produced a final answer, that is, we require that for any state Out, Is*P*Done(Out) returns either True or False in a finite amount of time to indicate, respectively, that Out be regarded as a target or a nontarget state.[13]

Now let's talk some more about the rules, or software, that govern a Turing machine. As I said above, the high-level software for a Turing machine is a lookup table that supplies a response triple for each possible stimulus pair. And the low-level software for a Turing machine forces it to cycle through the following three steps:

- (Turing A) The machine reads the symbol that is in the active cell. It combines the read symbol with its current state to make a stimulus pair (*internal state, read symbol*).
- (Turing B) Given the stimulus pair (*internal state, read symbol*), the machine looks in its high-level software to locate a corresponding response triple (*write symbol, move direction, new state*).
- (Turing C) On the basis of the response triple, the machine writes a symbol in the active cell, moves the head one step to the left or to the right, and enters a new state. If the machine is not in the halted state, it returns to step (Turing A).

One of Turing's great insights was that we can put the lookup tables for Turing machines down onto the tape along with the input data. That is, instead of running machine *M* on the data *d,* you can code *M* as a string of symbols *m,* and write the *m* pattern on the tape next to the data *d* to get a tape that we'll call *md*. And then a fairly routine bit of mathematical legerdemain can conjure up a specific universal Turing machine *U* such that the action of *U* on the tape *md* emulates the action of *M* on the tape *d*.

Note the exact analogy to the fact that, if *U* is a personal computer and *M* is some other personal computer, we can find an emulation program *m* so that the action of *U* on *md* is the same as the action of *M* on *d*.

Although every PC is universal, only some Turing machines are universal. All PCs are, after all, of a fairly high degree of complexity. But Turing machines can be made arbitrarily simple. Over the years there's been something of a competition among computer scientists to discover the simplest possible universal Turing machine. The simplicity of a Turing machine is gauged in terms of how many internal states the machine has and how many tape symbols it uses. The most recent record-holder, discovered by Stephen Wolfram and Matthew Szudzik on the basis of work by Matthew Cook, uses two states and five symbols. This means that the machine itself has an exceedingly simple lookup table. With two states and five symbols, there are only ten possible combinations of (*internal state, read symbol*), so the Turing machine's entire lookup table has only ten lines. Yet, by preparing the input tape in a suitable way, we can get this machine to emulate any possible computation.

Encouraged by this and some similar kinds of research, Wolfram conjectures in *A New Kind of Science* that universal computations are ubiquitous. This follows from his Principle of Computational Equivalence, or PCE, which I introduced a bit earlier in this chapter.

- *Principle of Computational Equivalence (PCE).* Almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication.

Let's delve into this more deeply than before.

The "almost all" at the start is so Wolfram can cover himself from a certain pointed criticism. The criticism stems from the fact, known since the 1960s, that there are in fact some gnarly class four Turing machines that aren't universal. But Wolfram's feeling is that, at least in nature if not in mathematics, such computations will be exceedingly rare. We might reasonably replace the phrasing "almost all" by "most naturally occurring."

When he speaks of an "obviously simple" process, Wolfram has class one and class two computations in mind. Recall that the class one computations run for a while and then enter a fixed state. There are actually two ways that a computation can be class two. On the one hand, it might go into a loop and begin precisely repeating itself. Or, on the other hand, the computation might generate a growing, orderly, unsurprising pattern. The three-state Turing machine depicted earlier in this section is an example of this style class two

computation. It doesn't exactly repeat itself, but what it's doing is "obviously simple."

The non-obviously-simple computations would be the disorderly class three computations and the gnarly class four computations. The disorderly computations seethe in a seemingly random fashion, and the gnarly ones generate intricate patterns.

What does Wolfram mean by two computations being "of equivalent sophistication"? We might take this to mean that they can emulate each other.

Note that if U is universal and if M can emulate U, then M must be universal as well. Consider an analogy: If you can imitate the actor Jim Carrey, who can imitate *anyone,* then you yourself can imitate anyone. To imitate Elvis, for instance, you imitate Jim Carrey imitating Elvis.

Given that we know that universal computations exist, if we take "of equivalent sophistication" to mean "able to emulate each other," we might phrase the PCE as follows.

- *Principle of Computational Equivalence, Second Form (PCE2).* Most naturally occurring complex computations are universal.

As I mentioned earlier, Wolfram advocates a related but distinct principle as well, the Principle of Computational Unpredictability.

- *Principle of Computational Unpredictability (PCU).* Most naturally occurring complex computations are unpredictable.

The PCE and PCU were to some extent inspired by Wolfram's searches over vast classes of Turing machines and other simple kinds of idealized computation. Wolfram's daring is to insist that his insights apply to *all* kinds of computations. In the chapters to come, we'll consider what the PCE and PCU might tell us about our world.

1.6: Inside the Beige Box

In this section we'll talk about real computers, that is, personal computers. There's no real need to talk about "supercomputers." Last year's supercomputer is next year's desktop machine.

Personal computers all have the same basic design: a processor and memory.

The processor is something like the head of a Turing machine, and the memory is like a Turing machine tape. Or, again, the processor is like a human reckoner, and the memory is like a sheet of paper.

The memory, often called RAM for random access memory, can be imagined as a long ribbon of cells. The PC's memory cells hold so-called words of memory. Here *word* does not mean "meaningful language unit." It simply means a particular fixed number of bits, let's say thirty-two zeroes or ones. Each word of memory has an address, and the memory addresses run from zero on through the thousands, millions, and billions, depending on how much RAM the particular machine has. The "random access" aspect of the memory has to do with the fact that the processor is easily able to read or write the contents of a cell at any desired address.

Let's look at what happens when a stored program architecture computer runs. The basic operation is for the processor to alternate between the following two steps:

- (Computer A) Fetch an instruction from memory.
- (Computer B) Interpret and execute the latest instruction.

The processor uses an address called the *instruction pointer* to keep track of which word of memory the processor is currently supposed to fetch. And it also keeps a *data read pointer* and a *data write pointer* to keep track of which memory slot to use for, respectively, reading or writing bits (see figure 15).

All of these pointers are stored in so-called registers that live right in the silicon of the processor. The processor has a few dozen such registers and they can be thought of as constituting part of its internal state.

According to which word the processor finds at the address of its instruction pointer, it will do one of the following:

- Read data from memory.
- Carry out logical or arithmetical operations such as AND or PLUS, and store the results in a "scratch-paper" register.
- Write data to memory.

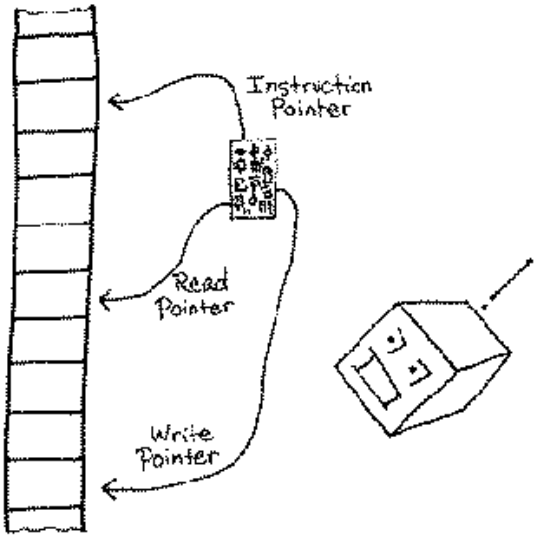


Figure 15:
Processor and Memory Ribbon

After fetching and executing each successive instruction, the processor will normally increment the instruction pointer to the next memory position, but certain instructions will tell it to override this default behavior and execute the following, fourth kind of primitive operation:

- Jump the instruction pointer to a new position.

Unlike a Turing machine's head, a personal computer's instruction pointer can hop to anywhere in memory in a single step. If you have some familiarity with programming, you may know that jumps in the instruction pointer's position can be caused by if-then-else statements, by loops, and by calls to procedures. The instruction pointer does a dance of computation.

A higher-level way to think of the difference between PCs and Turing machines would be to say that at any given time, a PC processor can access any memory location, whereas a Turing machine processor (or head) can only access one memory location. We represent this in the two diagrams in figure 16. In each diagram, the circle represents the processor and the row of cells

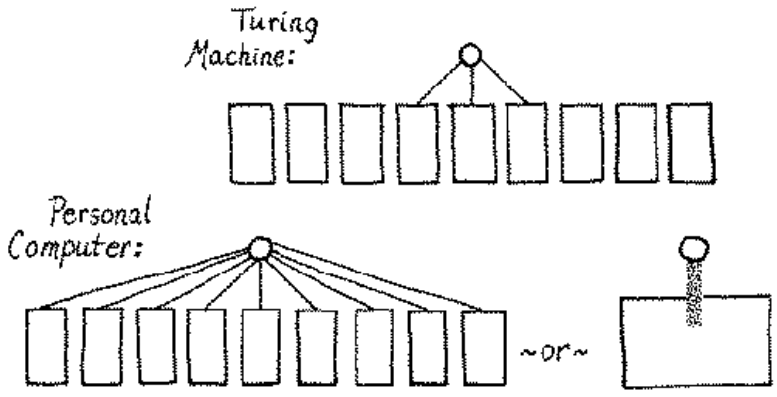


Figure 16: Architectures of Turing Machine vs. Personal Computer

We draw the Turing machine processor as having access not only to the current memory cell but to the two neighboring cells; this is because the processor is able to execute a "move left" or a "move right" instruction. We can draw the personal computer's architecture as a long row of memory cells, indicating the processor's global access by drawing connecting lines from the processor to each cell; or we can simplify by drawing the memory as a somewhat larger block meant to include lots of data, and drawing the processor-to-memory access line as thick, fuzzy, and gray, with the understanding that this kind of line means that the processor has rapid access to every nook of the associated memory box.

represents the memory. Computer scientists would say that the Turing machine has *local* memory access, while the PC has *global* memory access.

When the PC processor carries out logical and arithmetic operations, it manipulates the bits in the registers, often by combining one register's bits with the bits in another. More precisely, logic and arithmetic instructions may copy register values among each other, add register values, compare register values, and more. The actual execution of additions, multiplications, logical combinations, and so on, is handled by specialized circuitry on the chip, or what's sometimes called the *chip architecture* (there's that word again).

What about interactive inputs? Input devices can place a few bits or even a long patch of bits directly into the RAM. A keyboard feeds in perhaps thirty-two bits of data with each key press, while a disk drive can load in millions of bits at a time. Each time you move your mouse, the mouse, too, puts bits describing its clicks and moves into the computer memory. A program can go and check this area every so often, and in this way respond to the inputs.

Output devices convert bits into audible or visible display. A crude text screen might show a few hundred characters, using sixteen bits per character, whereas a graphics screen might display millions of colored pixels, with perhaps thirty-two bits of color code per pixel. You can print out your screens, or you can write the information onto a disk. A sound card converts swatches of bits into voices, music, and noise.

How is it that PCs often seem to be doing several things at once? Behind the scenes the machine allocates successive time-slices to a series of tasks and rapidly cycles around and around this task loop, giving the illusion that all the tasks are being worked on at once. In this fashion a PC can emulate a so-called parallel computer, which independently runs many computational threads at the same time.

Being a universal computer is empowering. It turns out that no matter what its particular architecture is, a universal computer can emulate *any* other computer, of *any* possible architecture. This isn't an obvious fact, nor is it something that's been formally proved—it's more in the nature of an empirical principle that's been deduced from the body of theoretical and practical knowledge of computation that we've accumulated. The principle is sometimes called Church's Thesis. We might phrase it like this:

- *Church's Thesis.* Any possible computation can be emulated by a personal computer with sufficiently large memory resources.

Alonzo Church proposed his thesis back in the 1930s, after observing that several different ways of defining computations were all equivalent to one another. The thesis becomes controversial when universal automatists argue that PCs can emulate naturally occurring physical processes—even with the understanding that the emulations will normally be unfeasible. The issue is that if physics were to involve *infinitely precise* continuous quantities changing according to exact laws, then the finitely complex digital electronic computers might not be able to adequately emulate physics. The resolution, which I'll discuss in CHAPTER TWO: *Our Rich World,* is to say that the quantities used in physics really have only a finite level of precision.

1.7: Plugged In

In the early 1980s, the science-fiction writer William Gibson coined the great word *cyberspace,* which now has come to mean, approximately, the Web. Originally the word also connoted virtual reality, in the sense of an immersive and shared graphical world.

In 1988, John Walker, then the chairman at Autodesk, Inc., of Sausalito, had the idea of building a software toolkit for creating shared virtual realities. Autodesk trademarked the word *Cyberspace* for an (unsuccessful) product called the Cyberspace Developer's Kit. William Gibson was somewhat annoyed by this and jokingly claimed he was going to trademark *Eric Gullichsen,* this being the name of the first lead programmer on the Autodesk Cyberspace project. I myself was employed by Autodesk at the time, recruited by Walker himself. I was helping to design and code a series of popular science software packages, including *Rudy Rucker's Cellular Automata Laboratory, James Gleick's Chaos: The Software,* and *Artificial Life Lab* (all of which are available for free download from this book's Web site, www.rudyrucker.com/lifebox/). I also helped write some demos for the Autodesk Cyberspace project, most memorably a lively flock of polyhedra that would circle the user's head.

Before we managed to get electronically linked multiple users into our cyberspace at the same time, Autodesk's stock price went down and I was out of the industry and back in the groves of academe, teaching computer science at San Jose State and writing a novel called *The Hacker and the Ants* about my experiences at Autodesk.

> What was cyberspace? Where did it come from? Cyberspace had oozed out of the world's computers like stage-magic fog. Cyberspace was an alternate reality, it was the huge interconnected computation that was being collectively run by planet Earth's computers around the clock. Cyberspace was the information network, but more than the Web, cyberspace was a shared vision of the Web as a physical space.[14]

Living through the dot-com boom in Silicon Valley was a trip; for a while there, money was growing on trees. I remember when a student in my Java

course came by the office to show me a job offer he'd gotten. They were offering him a fat salary, perhaps 30 percent more than what a humble professor makes. And he was wondering if he should ask for more! He was clever and likable, although disorganized and perhaps a little lazy. He got so excited about his impending career that he didn't hand in one of the class projects, which brought his average down to the C level, but it didn't matter; the bubble wanted everyone it could get, at least for a short time. I had thought he might be unemployed by now, or running an offshore coding group in Bangalore, but the other day he turned up again, having authored some software for anonymous Web-surfing, and still very much in the game.

The Web is here to stay.

When you push aside the hype and the biz buzz, the Web consists primarily of our personal computers, with the added feature that they can exchange data. When one computer gets information from another, we speak of them as a client and a server, respectively. The client is said to download files from the server, and, in the reverse direction, the client uploads files to the server so that other clients can see them.

A given PC may act as both client and server; indeed, in some local networks, all machines play both roles. It's more common, however, to have certain dedicated machines that function primarily as servers. These server machines are the same kinds of PCs that you might have at home, with the difference that dedicated servers usually use a Unix-type operating system. The clients and servers connect to one another via a hierarchy of machines called switches and routers, as indicated in figure 17.

My son Rudy Jr. runs what may be the only independent Internet service provider in San Francisco, www.monkeybrains.net. He keeps his machines in a cage that he rents for them in a so-called server hotel in a rough neighborhood. A robot flophouse. The server hotel was once a Macy's warehouse and is located next to a train track. Nearly all server hotels are next to train tracks so that their routers' fiber optic cables can follow the railway's right of way to the next server hotel down the line. The server hotel, or data center, holds three highly air-conditioned floors of wire cages, each cage stuffed with the machines of some stalwart Web entrepreneur.

Rudy's cage holds seventeen server machines and a router. The last time I visited, he pulled a plug out of the back of his router box and told me to look

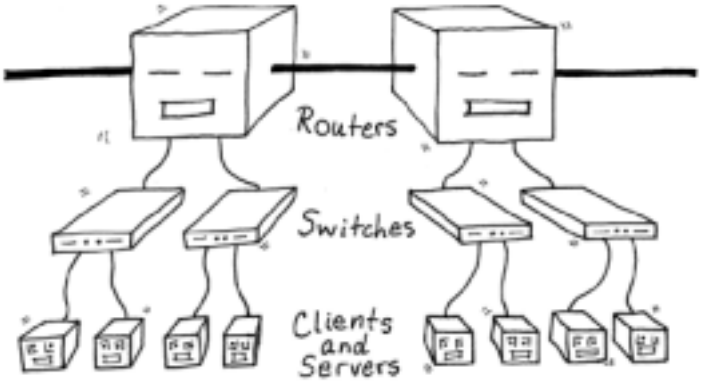


Figure 17: Networked Personal Computers

The physical details of the Web are more complicated than the figure shows. But a basic way of thinking of it is that individual server and client PCs connect to machines called switches that in turn connect to routers. Routers across the world are strung together via optical fiber connections; the network of linked routers is what you might think of as the Internet's backbone.

into the end of the wire. I saw a faint red light, grainy with information. It was his router's optical fiber line. "That's the color of the Internet," said my son. "You just saw a gigabit per second." Any information going through Rudy's router at that moment suffered a glitch, but the protocols of the Internet are smart enough to correct things like that.

To get a picture of how the Web works, let's step through an example. Suppose you enter my book's Web site address into your browser's address bar: www.rudyrucker.com/lifebox/.

The following sequence of actions results (leaving out numerous fiddling details):

- Your machine sends a message to Rudy Jr.'s server machine in San Francisco, stating your machine's name and requesting the page www.rudyrucker.com/lifebox/index.html.
- Rudy's machine sends bits describing this Web page to your machine.
- Your machine's browser software converts the bits into an image on your screen.

The transaction doesn't have to be one-way. My book's Web site has a guest book page.

Once the guest book page is showing on your machine, you can type something, press Enter, and your words will now be stored on Rudy's server machine. The next person to access the guest book page can see what you wrote there. The extra steps are these:

- Your machine sends bits to Rudy's machine.
- Rudy's machine incorporates your changes into one of its Web page files.

With a little more Web experience, you can do more than write things into someone's guest book: You can post images, maintain an online blog—or even establish your own Web site.

As I mentioned above, when your machine reads in some information across the Web, this is a *download,* and when your machine writes some information into some other location on the Web this is an *upload.* Be warned that some people use these words the opposite way around. But as John Walker convincingly puts it, "When you offer your data to the great Net God like the smoke of burnt offerings rising into the heavens—this is an *upload.* And when the riches of the Web rain upon you like manna—this is a *download.*"

The Web greatly leverages your access to information by means of hyperlinks. When you click on a hyperlink on a Web page, the server machine sends your machine the name of a new machine, your machine contacts the new machine and asks for the page, and the new machine sends your machine the new page. For you, it's as if the Web is a seamless whole, and it doesn't make all that much difference which server you initially connect yourself to.

Can we think of Web itself as a kind of computation? Sure. As long as something is rule-based it's a computation. And the Web has rule-based behavior—messages dart back and forth, requesting and delivering data. The initial input is the machines and their connections, and the interactive input is the requests emanating from the machines. The behavior of the Web in and of itself is thoroughly deterministic. Even when a message needs to make a seemingly random choice among several equally good paths, a deterministic pseudorandom

algorithm is in fact used to make the decision. And the data requests made by human users are additional interactive inputs to the system.

We could also imagine a completely deterministic Web in which the client requests are being generated by programs running on the individual client and server machines. Web crawlers are examples of this kind of automated Web surfing. A client running a Web crawler will successively visit one page after another, accumulating information on what it finds. A search engine like the currently popular Google uses Web crawlers to produce information for its own large database. When a user goes to the Google site and asks for pages relating to a given topic, the Google software uses its Web-crawler-built database to suggest links. As an additional wrinkle, Google ranks each page, using criteria such as how many other pages have links to the given page.

Let's think a bit more about the Web as a computer. Generalized rule-based systems—computers in the broad sense of the word—can be based on a wide range of underlying architectures. That is, the mutual interactions of a computer's hardware, software, and data can be organized in many different ways. A computer's strengths and weaknesses have much to do with its architecture. Three commonly seen architectures are the serial, the networked, and the parallel. A PC has a serial architecture, in which a single processor has global access to a single memory set. Classical physics, on the other hand, can be thought of as a parallel architecture, in which many processors have local access to a single shared memory set (the world).

A *network architecture* has five distinctive characteristics. The first three are these:

- Many distinct processes.
- Each process is associated with its own private block of memory.
- The processes can access one another's memories by exchanging read and write requests.

The Web has many processors, each of which has its own private memory. A Web-linked machine has instant access to any location of its own memory, but it has only an indirect access to the memories of the other machines.

The tree structure in our first drawing of the Web (figure 17) was an implementation detail. The essence of the network architecture appears in figure 18.

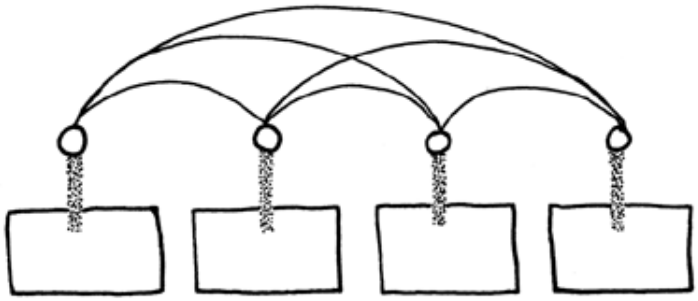


Figure 18: Network Architecture

The circles are the computer processors and the boxes are the memory sets of the individual machines. The arced lines at the top are connections by which the processors make read-write requests, while the thick gray lines below represent the fact that each processor has full rapid access to its own private memory block.

We speak of each processor-memory combination as a *node.* In terms of the figure, a node is a circle plus its associated rectangle of memory.

We can describe the memory access as follows. In order to read memory from another node, a given node needs to send a request to the remote node's processor and wait for this node to retrieve and send the desired information. Writing to the memory of another node requires a similar procedure, involving a similar kind of request. An important characteristic of the networked architectures is that a given node can deny these requests.

- A network node may deny incoming read or write requests.

Another characteristic feature of the network architecture is the lack of any kind of systemwide synchronization. Indeed, networks are often called *asynchronous.*

- Each network node sends and processes requests according to its own schedule and speed.

The network architecture is found in several naturally occurring forms. A living organism can be thought of as a network whose individual processors

are the organism's cells. And our society is a network in which the processors are human beings. In both cases each individual processor has its own private memory, the processors share data by exchanging signals with one another, and the processors can refuse requests.

A rude question. If the Web is a computation, then what the heck is it computing? The easy answer is that computations don't have to be "about" anything. They can just occur. Rain running down a windowpane isn't about anything, but certainly there's an intricate computation taking place.

Certainly it would be interesting if the Web really were somehow computing something deep. The hooked-together computers of the Web are at least superficially reminiscent of the coupled neurons that make up a human brain. Could the Web ever act as a planetary mind? This question is a variant of the old question of whether human society as a whole has a group mind. I think that in both cases the answer is a qualified yes—I'll say more about this in CHAPTER FIVE: *The Human Hive.*

1.8: Flickercladding

A cellular automaton (CA for short) is a parallel computation that operates on a memory space that is a one-, two-, three-, or higher-dimensional grid of cells. The memory can be, for instance, a one-dimensional tape like a Turing machine's tape, a two-dimensional grid of cells like a reckoner's paper, or a lattice of three-dimensional cubes.

Each cell has its own associated processor, and each cell contains a small amount of data called its *value.* As we'll see in CHAPTER TWO: *Our Rich World,* when modeling physics, we turn to CAs in which each cell value consists of one or several real numbers. But often we focus on discrete-valued CAs, that is, CAs whose cell values are a single integer or even a single bit.

We depict the architecture of one-dimensional and two-dimensional CA as in figure 19, where the processors are circles attached to square memory cells.

The CA computation proceeds in discrete steps. At each step, every cell is simultaneously updated. How is an individual cell updated? Each cell processor has a rule that computes the cell's new value based upon the cell's current value and the values of a few neighboring cells. In implementing the flow of heat as a CA, for instance, the rule might simply be to average a cell's temperature value with the temperature values of the cells adjacent to it.

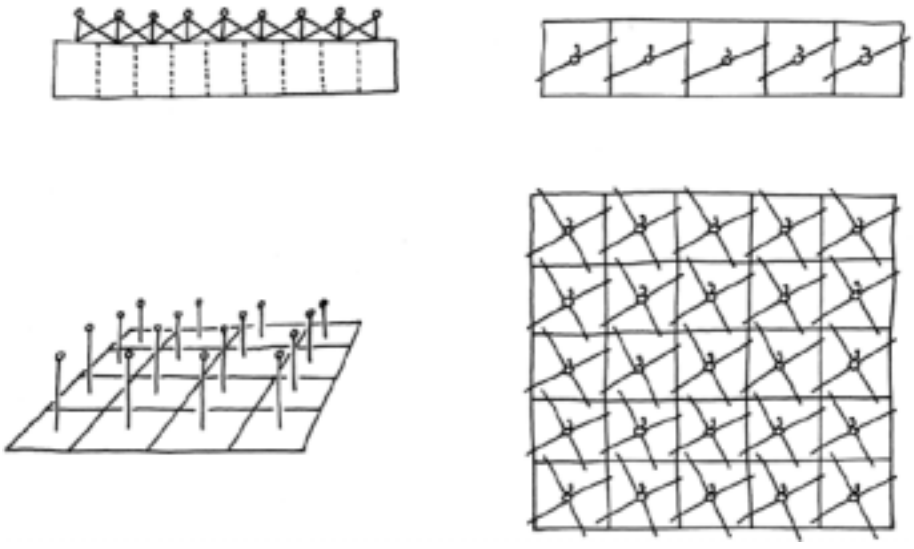


Figure 19: Architecture of One- and Two-Dimensional CAs

The top row shows two images of a one-dimensional CA architecture, and the bottom row shows two images of a two-dimensional CA architecture. In the left column we draw the processors as floating above the cells, and in the right column we draw them down inside the cells. The lower left image would be more accurate if each processor had lines coming down to the neighbor cells.

Although we could in principle use different update rules for the different individual cells, it's more common to study CAs in which all the cells use the same rule and look at the same pattern of nearest neighbors.

In short, CAs are defined so as to satisfy these five conditions:

- *Many processors.* A CA has one processor per memory cell.
- *One shared memory.* The cells are arranged into a single memory grid.
- *Locality.* The CA update rules are local, that is, a given cell's new value depends only on the present values of the cells in some fixed neighborhood of the cell.
- *Homogeneity.* Each CA has the same update rule.
- *Synchronization.* All of the CA cells are updated at once.

Cellular automata seem to have been invented in the late 1940s at the Los Alamos, New Mexico, laboratories by Stanislaw Ulam and John von Neumann. Both these men were primarily mathematicians, but their interests had exceedingly wide range. Recall that von Neumann was instrumental in the creation of the first electronic computers. He also did work on theories of infinity, the foundations of quantum mechanics, economics, and game theory.

Ulam, too, did work on theories of infinity, inventing what stood for many years as the largest kinds of numbers anyone could dream up: the so-called measurable cardinals. He was involved in computers as well, using a machine called MANIAC to come up with some novel methods of simulating nonlinear physics (see figure 20). And, with Edward Teller, Ulam was the co-inventor of the hydrogen bomb.

Ulam's first published reference to cellular automata appeared around 1950, at the time he was helping von Neumann design a self-reproducing machine.[15] Ulam carried out some investigations of discrete-valued CAs and then, in the 1950s, he switched his attention to continuous-valued CAs, that is, cellular automata in which the values are real numbers—this work we'll discuss in chapter 2.

Figure 20: Stanislaw Ulam Demonstrating the MANIAC Computer

The little girl is Ulam's daughter Claire. I discovered this picture in S. M. Ulam, Adventures of a Mathematician *(Berkeley: University of California Press, 1991).*

CAs didn't really catch on until 1970 when, in his popular "Mathematical Games" column for *Scientific American,* Martin Gardner wrote about how John Horton Conway, a mathematician at the University of Cambridge, had discovered a two-dimensional CA so rich in patterns and behavior that it was known as the Game of Life, or simply Life.

In Life each cell value consists of a single 0 or 1 bit, indicating if the cell is "dead" or "alive." Each cell's processor

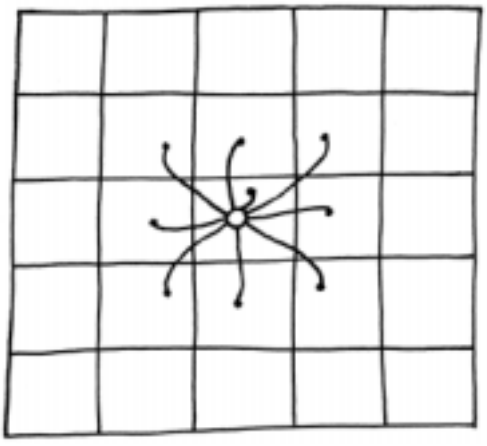


Figure 21:
A Cell Neighborhood in Conway's Life

Note that this notion of neighborhood differs from the style of two-dimensional cell neighborhood we drew in figure 19—where the two-dimensional cell processors were only looking at five cells each.

looks at nine of the memory cells, the 3×3 neighborhood around the cell (figure 21).

If we speak of the cells as being alive or dead, we can describe the Game of Life rule in the following colorful fashion:

- If a dead cell has exactly three live neighbors, they spawn into the cell and it, too, becomes alive. Otherwise a dead cell stays dead.
- If a live cell has exactly two or three live neighbors other than itself, then it stays alive; otherwise it dies of loneliness or overcrowding.

Conway's vague initial goal had been to find a cellular automaton rule in which simple patterns could grow to a large size, but he doubted if any patterns could grow forever. Gardner proposed this as a challenge problem:

> Conway conjectures that no pattern can grow without limit. Put another way, any configuration with a finite number of live cells cannot

> grow beyond a finite upper limit to the number of live cells on the field. This is probably the deepest and most difficult question posed by the game. Conway has offered a prize of $50 to the first person who can prove or disprove the conjecture before the end of the year. One way to disprove it would be to discover patterns that keep adding live cells to the field: a "gun" (a configuration that repeatedly shoots out moving objects such as the "glider"), or a "puffer train" (a configuration that moves about and leaves behind a trail of "smoke").[16]

The prize was won a month later by William Gosper and five fellow hackers at MIT; legend has it that they did an automated search. They sent Martin Gardner a telegram with the coordinates of the cells to turn on to make a glider gun, depicted in figure 22.

Steven Levy's *Hackers* has a good section about Gosper and the early excitement over Life among the users of the PDP-6 computer at the MIT

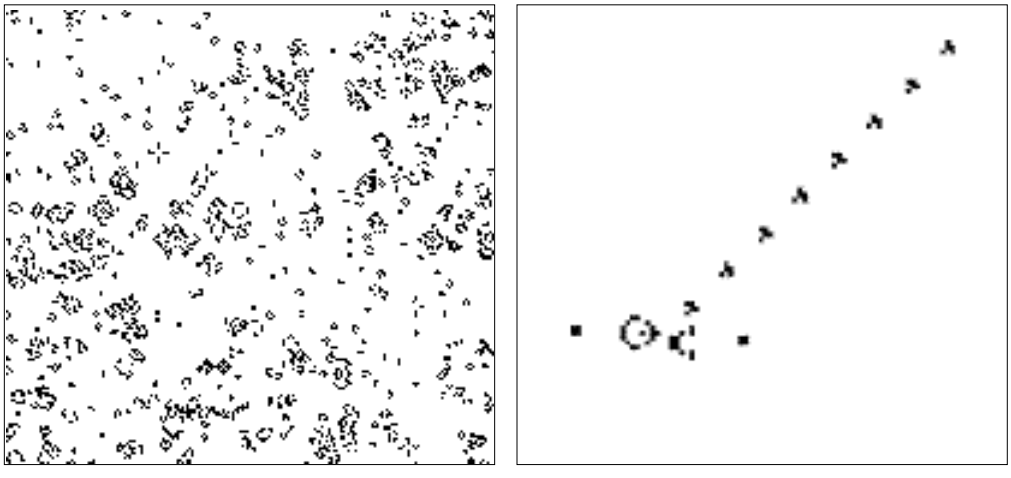


Figure 22: A Life CA Soup and the Glider Gun

The left-hand image shows Life running on a randomly seeded pattern. This kind of start almost always ends with a few static blocks and small oscillators. The right-hand image shows Gosper's glider gun sending out a stream of gliders. The gliders continually move toward the upper right. What happens at the edges? In simpler CA simulations the edges are wrapped around like in an old videogame—if a glider moves off the top edge, then it comes back from the bottom edge, and so on. But more sophisticated setups may model additional off-screen cells as well..

Artificial Intelligence Project. Levy quotes Gosper, telling how he saw Life as a way to

> basically do new science in a universe where all the smart guys haven't already nixed you out two or three hundred years ago. It's your life story if you're a mathematician: every time you discover something neat, you discover that Gauss or Newton knew it in his crib. With Life you're the first guy there, and there's always fun stuff going on. You can do everything from recursive function theory to animal husbandry. There's a community of people who are sharing their experiences with you. And there's the sense of connection between you and the environment. The idea of where's the boundary of a computer. Where does the computer leave off and the environment begin?[17]

One must remember that 1970 was still the Dark Ages of computing; Conway himself ran his Life simulations by marking the cells with checkers or flat Othello counters. For Gosper and his team to get Life to run on a monitor at all was a nontrivial feat of hacking—it was a new thing to do with a computer. After Gardner's second column on Life, the game became something of a mania among computer users. By 1974, an article about Life in *Time* could complain that "millions of dollars in valuable computer time may have already been wasted by the game's growing horde of fanatics."[18]

More and more intricate Life patterns were found all through the 1970s, and by 1980, Conway and his colleagues had enough Life machinery at hand to sketch a proof that Life can be used to simulate any digital computation whatsoever, that is, a CA running the Life rule is a universal computer.[19]

A number of people at MIT began studying CAs other than Life during the 1970s. One the most influential figures there was Edward Fredkin. Although he himself held no higher degrees, Fredkin was a professor associated with the MIT Laboratory for Computer Science, and he directed a number of dissertations on CAs.

Fredkin envisioned a new science where we represent all physical quantities as packets of information. The substrate on which these packets move was to be a CA. Not to put too fine a point on it, Fredkin argued that,

at some deep level, the world we live in is a huge cellular automaton. Although Conway had already expressed opinions to the effect that in a cosmically large Life simulation one might see the evolution of persistent patterns that are as intelligent as us, Fredkin was the first to suggest that the world we live in really *is* a CA.[20] He was thus one of the first to espouse universal automatism—although Fredkin prefers to name his view *digital philosophy*.

Fredkin formed the Information Mechanics Group at MIT along with Tommaso Toffoli, Norman Margolus, and Gerard Vichniac. Working together, Margolus and Toffoli built the so-called CAM-6 cellular automaton machine in 1984, a board that you could plug into the early-model IBM personal computers so as to see CAs running at a rapid clip.

Also in the 1980s, Stephen Wolfram became interested in getting an exhaustive overview of what kinds of CA computations are possible. In order to limit the number of possible rules, he started with very simplest CAs, in which the cell space is a one-dimensional row of cells, the possible cell states are zero and one, and each cell "sees" only itself and its nearest neighbors on the left and on the right, making a three-cell neighborhood. A CA of this simple kind can be specified by describing how a cell's new value depends on which of the eight possible three-cell neighborhood configurations it lies in. This makes for 2^8, or 256, possible rules, which are conventionally labeled by the integers from zero to 255.

Wolfram began by starting each of the basic 256 rules on a row full of randomly chosen zeros and ones and observing what classes of behavior occur. He found five general kinds of behavior. The distinctions extend to experiments where we start the rules on a simple row with but a single dark "one" cell. As suggested by the images in figure 23, Rule 254 "dies out" or becomes uniform, Rule 250 generates a checkered or periodic pattern, Rule 90 generates a recursively nested triangle pattern, the right-hand part of Rule 30's swath is random-looking, and Rule 110 has persistent local structures that move across the cell space and interact with one another.

Wolfram decided that a nested pattern was not really so different from a repeating pattern, and chose to group the CA behaviors into the four classes we mentioned earlier in this chapter.

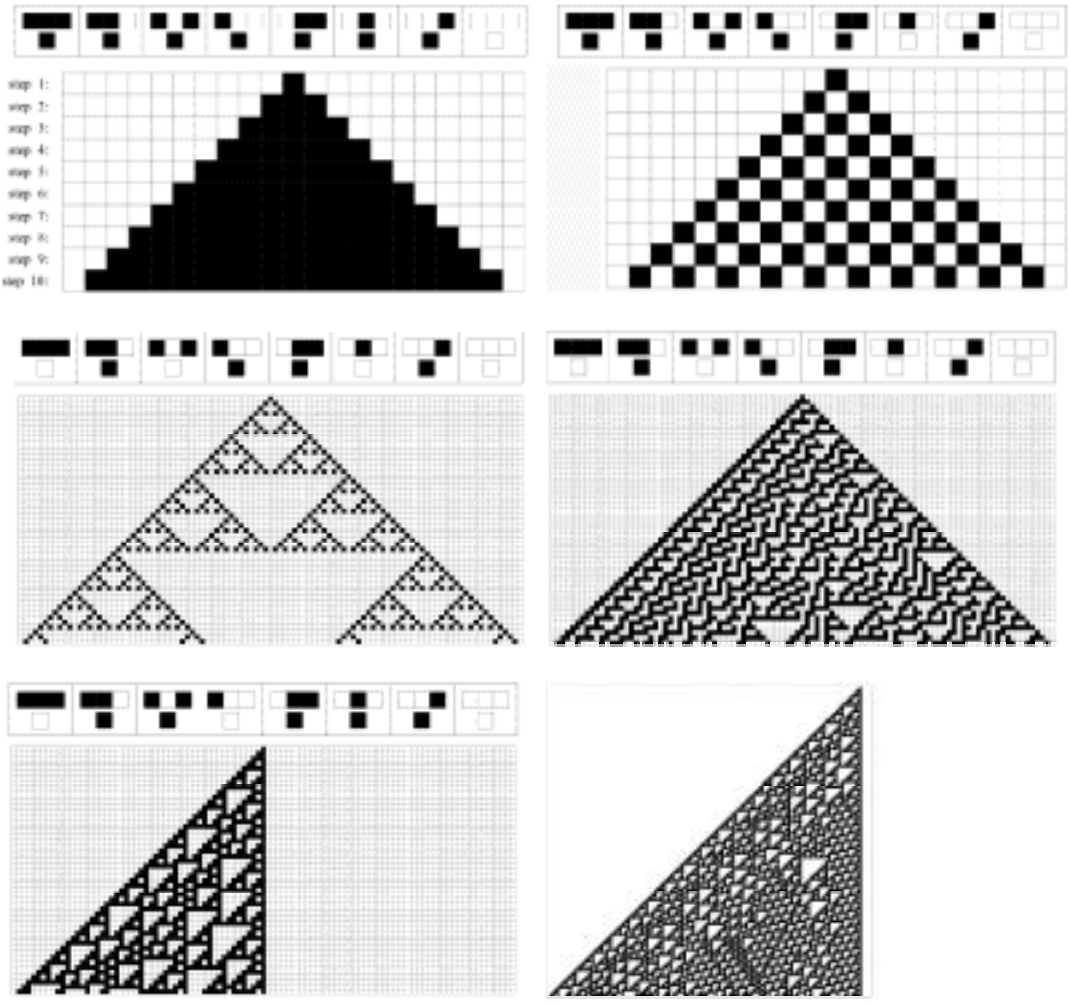


Figure 23: Five Kinds of CA Behavior

Left to right and top to bottom, these are Rules 254, 250, 90, 30, and two views of Rule 110. Wolfram views these rules as being of, respectively, computational class one, two, two, three, and four. Each image (except the sixth) contains a row of symbols describing the CA rule and a triangular pattern showing how the CA evolves over time. The rows of symbols show which new value of a cell is selected by that rule for each of the eight possible cell neighborhoods. (If you were to interpret the eight new-cell values as the digits of a binary number, by the way, you get the code number used to describe the given rule.) The triangular patterns show the successive states of the one-dimensional CA tape, with time running down the page. Note that each CA starts out with a single black cell. In order to give a better idea of Rule 110, we've added a zoomed-out view of its appearance later on.

- *Class one.* Dies out or becomes uniform.
- *Class two.* Becomes periodic or produces nested structures.
- *Class three.* Produces seething, seemingly random, patterns.
- *Class four.* Shows persistent local structures that move about.

The robustness of this classification is quite well supported by what one might call taxonomic studies of the kinds of computations that occur across a wide range of contexts. For instance, the same four classes of behavior appear if we look at more complicated CAs, such as those that allow more than two symbols, those that look at more than the very nearest neighbors, or those that use higher dimensional cell-spaces. And the same four classes can be found among the Turing machines.

Note that any universal computer can exhibit all four classes of computation. Depending on its input, it can produce *simple* (class one or class two) computations that die out or repeat, disorderly *random-looking* (class three) computations, or a purposeful-seeming *gnarly* (class four) computations.

I need to remark again that distinguishing between class three and class-four computations can be difficult. Wolfram's definitions of these notions are not formalized; they're more like empirical notions that have been formed from extensive observation. Note also that a periodic class two computation can look like a class three or class four computation if it takes a long time to get around to repeating itself, and even a class one computation can seem like a class three or a class four computation if it takes it a long time to die out.

The classes of computations generated by Conway's Life CA depend on the initial condition. The simplest Life patterns simply die off to a blank screen, which is class one behavior. A typical random seeding of a Life CA dies down to static blocks and oscillating patterns, which are class two. If only a central region of the world is seeded, a random Life start will in fact spew out a few gliders that head off into empty territory. Even if the gliders could indefinitely travel through empty cell space, if they're not interacting, then nothing interesting is happening and we still have only class two.

But, as mentioned above, Conway and some of his colleagues were eventually able to prove that Life is computation universal. This means that for any possible computation *M*, we can find a cell pattern so that Life seeded with this pattern will emulate *M*. So, since Life is universal, we know that it

can exhibit both class three and class four behavior. If Life emulates, say, the output of a some little algorithm usable as a random number generator, it will appear to be class three, while if it pulses out flocks of gliders grouped like the digits of pi, it will be class four.

What's the simplest possible universal CA rule? Stephen Wolfram and Matthew Cook were able to prove that the gnarly little CA Rule 110 is computation universal. The possible universality of the messy-looking Rule 30 remains, however, undecided. If Rule 30 proves to be nonuniversal, this would serve as a counterexample to the Principle of Computational Equivalence or PCE—for then Rule 30 would be an example of a complex computation that is not universal and is thus not as sophisticated as some other complex computations.

In 1984 Wolfram wrote a revolutionary article pointing out some fundamental similarities between physics and cellular automata.[21] He suggested that many physical processes that seem random are in fact the deterministic outcome of computations that are so convoluted that they cannot be compressed into shorter form and predicted in advance. He spoke of these computations as irreducible and cited CAs as good examples. His article included some intriguing color photographs of one-dimensional CAs.

Wolfram's article fascinated me so much that in April 1985 I set out to meet Wolfram, Margolus, Toffoli, and the other new cellular automatists, eventually writing an article on them that appeared in, of all places, *Isaac Asimov's Science Fiction Magazine.* The trip was something of a conversion experience for me, and in 1986 I left my career as a freelance writer for a job as a computer science professor at San Jose State University. CAs had permanently blown my mind.

One of the first things I started doing in my computer science classes was, of course, getting my students to write programs to display cellular automata. The computations were fairly intensive for the machines of that time, so we had to work hard to make the rules run fast, even to the point of programming them in low-level assembly language. One of my favorite early CA programs, when seeded a certain way, created images that looked like a continually mutating cyberpunk woman's face, as shown in figure 24. I called this "woman" Maxine Headroom after the then-popular TV show featuring a computer-animated character called Max Headroom.

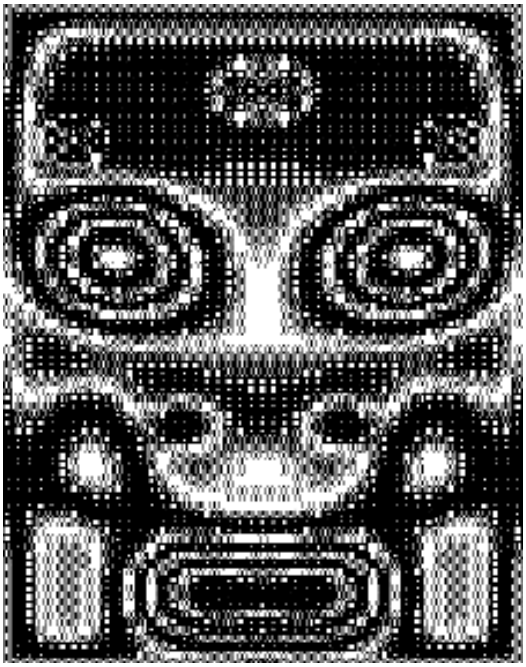


Figure 24: Maxine Headroom

This old image was created by a rule I call the Rug rule. Each cell holds an integer between zero and 255, and the update rule takes the average of a cell's neighbors, adds 1, and if the result is larger than 255, sets the value back to zero. This implementation is from the RC software component of CelLab, and actually uses obscure text characters for the graphics. To create Maxine's face, I start with a grid of a particular size—I think 43 × 80 in this case. (For full effect, I turn the monitor on its side so the shape is more like a face.) I freeze the (invisible) outer edges of the cell world at the maximum value and seed the interior with an egg-shaped pattern near the bottom. The upper part of the screen evolves into an elliptical forehead with a circular pair of eyes, the bottom of the screen produces an elliptical mouth, and the cells in between naturally shape themselves into the forms of brows, nose, and cheekbones.

The Maxine Headroom picture wouldn't mean much if it were based on something like a smoothing algorithm run on a seed image of a face. But it arises in a natural way, by a process that a biologist might call morphogenesis. Might it be that the forms of real people are simply patterns that are natural for growing masses of cells to form? I'll return to this question in CHAPTER THREE: *Life's Lovely Gnarl.*

I managed to get hold of one of Margolus and Toffoli's CAM-6 cellular automaton accelerator boards and found out how to make the board run. To make it work, you had to plug it into a certain early-model PC called the IBM XT. I mastered the board's arcane control language—a "reverse Polish" dialect known as Forth—and began writing programs.

Thus began one of the most exciting periods of my life. I became a cellular automata missionary, a Johnny Automataseed. I tracked down Bill Gosper in his office at the Symbolics Corporation in Palo Alto, California, and made him

look at new rules that went beyond his beloved Game of Life. As it happened, Gosper only had refrigerator-size computers, so I had to take the case off his secretary's IBM XT so I could plug in the CAM-6 board. He teased me for doing this, asking if I enjoyed "playing dentist," but the colorful demos of the new rules intrigued him.

Before I even heard about cellular automata, I'd begun writing my *Ware* series of novels about autonomous robots living on the moon. I'd always disliked how dull robots looked in science-fiction movies—like file cabinets or toasters. So I'd taken to decorating my fictional robots' bodies with a light-emitting substance I dubbed *flickercladding*. My original inspiration for flickercladding was the banks of flashing lights that used to decorate the sides of mainframe computers—signals reflecting the bits of the machines' changing internal states. As I imagined it, "The color pulses of the flickercladding served to emphasize or comment on the robots' digital transmissions; much as people's smiles and grimaces add analog meaning to what they say."[22]

My flickercladding wasn't meant to display blocks of solid hues, mind you; it was supposed to be fizzy and filled with patterns. And when I encountered CAs, I recognized what I'd been imagining all along. Reality had caught up with me.

This was also the period when, accompanied by my science-fiction hero Robert Sheckley, I went to Tim Leary's house in Hollywood and took—acid? No. Took apart his IBM XT and jacked in my trusty CAM-6 board so that good Dr. Tim could get a taste of CAs and their real-time mandala flows and creepy-crawly life-forms. Tim instantly got the picture: Computation could be about light shows, about mind expansion, about having fun. What a wonderful day that was.

What were Sheckley and I doing at Leary's house? Well, one of Sheckley's tummler friends was promoting a Hollywood pitch that Leary should host a weekly science TV show and that the Scheck-man and I should write his scripts. Too bad it didn't work out.

Soon after I demoed the CAs for Tim, I had the opportunity to bring my CAM-6-equipped PC to a 1987 conference called Hackers 3.0—keep in mind that back then "hacker" simply meant "fanatic programmer." As a relative novice to computing, I felt a little diffident joining this august geekly company, but they were most welcoming. With Silicon Valley just opening up, there seemed to be enough room for everyone.

It was a great conference for me. I did cellular automata demos all night long, and the hackers were blown away by the images. They pried the special-purpose CAM-6 board out of my machine, sniffed it over, and pronounced that it could readily be emulated by clever software.

As it happened, the demonically gifted John Walker of Autodesk fame was in the crowd, and he was just the man to carry out the sought-for hack. Within a year, I'd taken leave from my professor's job and started working for Autodesk, helping Walker create a fast all-software CA simulator known as *CelLab*.[23]

CelLab emulates the parallel CA architecture within the serial architecture of a personal computer—a reminder of the fact that any of our universal computational systems can simulate the behavior of any other. Using a different architecture doesn't affect what's *in principle* computable. But a system's architecture does have a lot do with what kinds of computations are *feasible* for that system.

In the 1990s, I became interested in continuous-valued CAs, in which the cells can contain one or several real numbers instead of simply holding simple integer values. Working with my students at San Jose State University, I designed a Windows program called CAPOW, which is very useful for investigating these kinds of CAs, and I continue working with CAPOW to this day.[24]

I might mention here that not all computer scientists like the idea of continuous-valued CAs. Digital computer programs commonly allow for some four billion different values for a continuous-valued real number. This means that whereas the Game of Life CA has two possible states per cell, a continuous-valued CA might have four billion possible states per cell. And if I use two real numbers per cell—as I often do—I'm looking at sixteen quadrillion possible cell states. Stephen Wolfram sometimes chides me for using complicated rules—he argues that it's more scientifically significant to discover an interesting behavior in the context of a system that's been pared down to be minimally complex.

But the physical world is anything but clean and simple. And even so, nature repeats the same structures over and over. The same simple patterns arise even when the computations become very intricate. You might say that Platonic forms are robust against scuzz. And these qualities of robustness and universality are worth modeling.

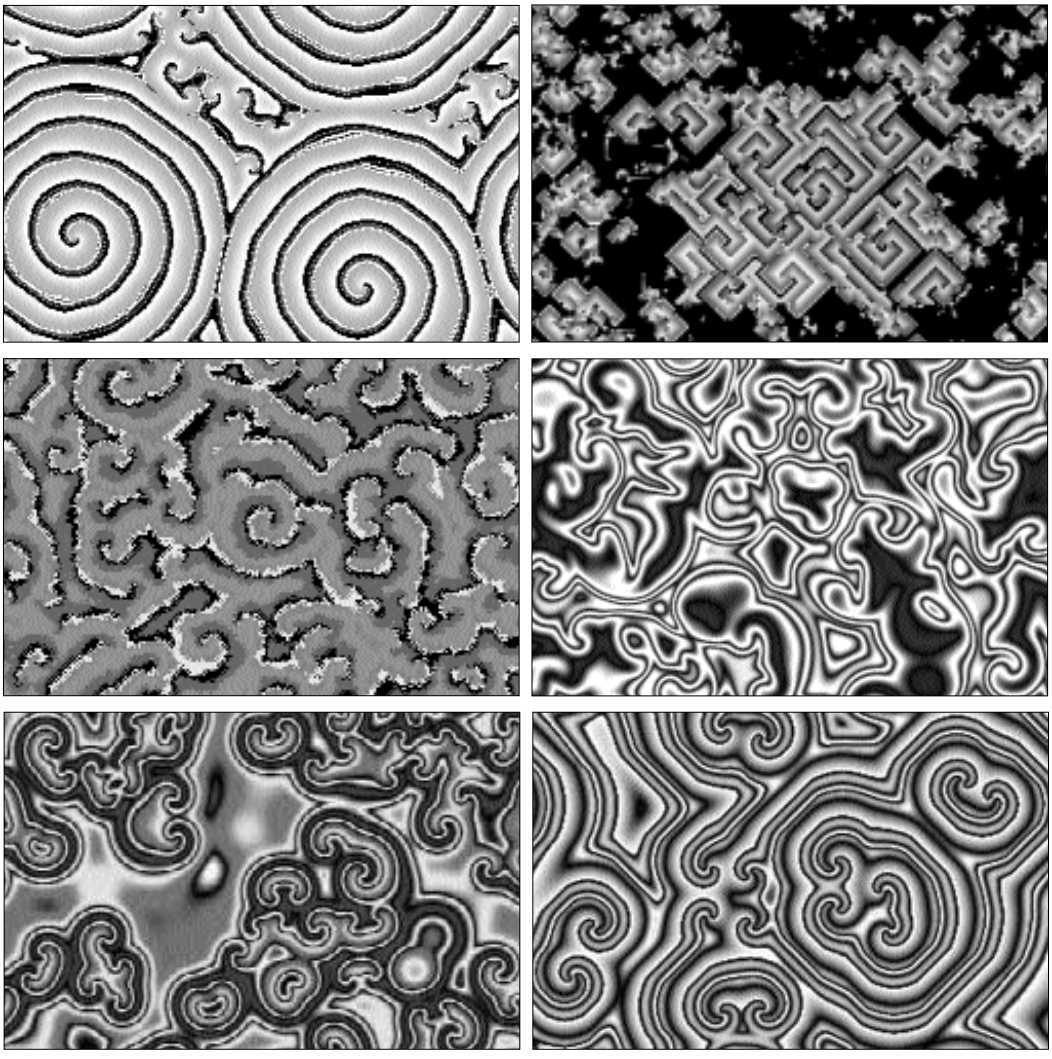


Figure 25: Cellular Automata Scrolls

I ran these rules at a resolution of 320 × 200 cells, letting the left edge wrap around to the right and the top wrap around to the bottom. The first three have states consisting of a single integer per cell, while the latter three have two real numbers per cell. All were started with a random initial pattern. Reading across the page and from top to bottom, the six images are: Gerhardt and Schuster's Hodgepodge rule, my RainZha rule, Toffoli and Margolus's Tubeworms rule, one of Hans Meinhardt's Activator Inhibitor rules, a Double Logistic predator-prey rule, and Arthur Winfree's Belousov-Zhabotinsky rule.[25]

Yes, it's important to find complex patterns in simply defined systems. But to my way of thinking it's significant that we can find exactly the same kinds of patterns—and no others—in the much more complicated systems.

By now I've looked at many thousands of two-dimensional CAs. As it turns out, scrolls are perhaps the most interesting new kinds of CA patterns that emerge when moving from one-dimensional to two-dimensional CAs—my favorite CAs are those that spontaneously generate scroll patterns from both orderly and disorderly start patterns. Scroll patterns occur in both simple and complicated rules and—in keeping with the point I was just making—the scroll-shaped forms in complicated CA systems can be just as clean and sharp as those defined by simpler rules.

A sampler of two-dimensional scroll CAs appears in figure 25.

Nature likes scrolls as well. Consider that, for instance, the cross section of a mushroom cap, cut from top to bottom, bends around like a scroll—while the entire cap forms a kind of three-dimensional scroll. Both a bean and a fetus resemble fleshy scrolls. Brain tissue is replete with three-dimensional scrolls. Candle or cigarette smoke inks the air with scrolls—and the pairs of eddies that form behind a moving spoon in a coffee cup are scrolls as well. Figure 26 shows a computer simulation of three-dimensional scrolls.

Figure 26: A Three-Dimensional CA with Scrolls

Here's a Hodgepodge-style CA running on a three-dimensional array of cells. The entire solid block is filled with interacting shapes like scrolls, mushroom caps, jellyfish, and whirlpools. The block wraps around, that is, patterns moving across one face continue into the opposite side of the cube.

What is the computational class of CA scrolls? Something that's a bit hard to capture in a printed picture is how dynamic they are. The scrolls are continually turning, with the pointed ends melting away as they approach the enfolding lines. The rules are certainly not class one. The strongly ordered patterns also preclude our calling them class three—these robust patterns are anything but random in appearance.

So we're left with deciding between class two and class four. If we run a scroll rule on a very small grid, we may find the rule filling the grid with a single monster scroll that executes a repetitive class two cycle. But if we give the rule room to grow several scrolls, then we seem to see class four behavior. A test case for this appears in figure 27, which shows eight stages of the so-called Hodgepodge rule, reading left to right and top to bottom. Notice that each image is subtly different and that, in particular, the diamond-shaped region in the center is not repeating itself.

If I let the simulation run, say, another thousand steps, I find that the whole general shape of the central pattern will have changed. So I'm prepared to claim that some of the scroll-generating CA rules are class four.

Within the framework of *The Lifebox, the Seashell, and the Soul,* this seemingly arcane observation turns out to be important. How so? Quite specifically, I'll be arguing that many of the computations involved in living organisms are of the same type that produce these unpredictable CA scrolls. There's even a possibility that our brains are in fact animated by electrochemical processes that behave like three-dimensional CA scrolls. Metaphorically speaking, that churning little central region of the Hodge patterns in figure 27 is like a brain working away inside a scrolly skull.

More generally, the free and unpredictable play of CA scrolls is a perfect example of how deterministic processes can produce gnarly shapes and interesting behaviors.

In this chapter we've talked about very explicitly computational processes: humans reckoning with numbers, Turing machines, personal computers, the Web, and cellular automata. And in the rest of the book we'll be viewing some less obviously deterministic processes as computations: physics, biology, the mind, and human society.

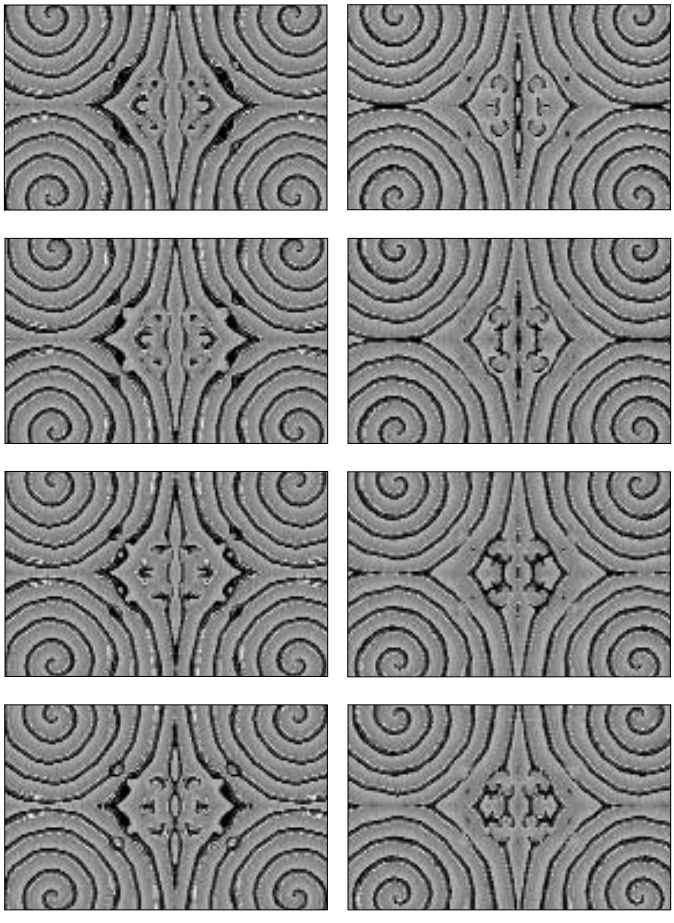


Figure 27:
Class Four Scrolls

For these pictures, I seeded the Hodgepodge rule with a small square in one corner, whose effects immediately wrapped around to the other corners. I let the rule run for about a week, passing through some fifty million updates. And then I paused and single-stepped the rule, capturing eight images, with each image five updates later than the one before.

THOUGHT EXPERIMENT TWO: THE MILLION CHAKRAS

Teaching her third yoga class of the day, Amy Hendrix felt light-headed and rubbery. She walked around, correcting people's poses, encouraging them to hold their positions longer than they usually did. Her mind kept wandering to the room she was hoping to rent. New to San Francisco, she'd been sleeping on couches for six weeks. But she still dreamed of becoming a force to be reckoned with in the city scene.

It was time for Savasana, the Corpse Pose, with everyone lying on their backs. Amy turned off her Tabla Beat CD and guided the closing meditation.

"Feel a slow wave of softness moving up your legs," she began. "Feet, calves, knees, thighs." Long pause. "Now focus on your perineum. Chakra one. Release any tension hiding there. Melt with the in-breath, bloom with the out. Almost like you're going to wet your pants." Amy occasionally added an earthy touch—which her mostly white clients readily accepted from their coffee-colored teacher.

"Gather the energy into a ball of light between your legs," continued Amy, pausing between each sentence, trying not to talk too much. "Slowly, slowly it passes upward, tracking your spine like a trolley. Now the light is in your sex chakra. Let it tingle, savor it, let it move on. The warmth flows through your belly and into your solar plexus. Your breath is waves on a beach."

She was sitting cross-legged at one end of the darkly lit room. The meditation was getting good to her. "Energy in, darkness out. The light comes into your chest. You're in the grass, looking at the leaves in a high summer tree. The sun shines through. Your heart is basking. You love the world. You love the practice. You love yourself. The light moves through your neck like toothpaste out a tube. Chakra five. The light is balancing your hormones, it's washing away your angry unsaid words." Pause. "And now your tape loops are gone."

She gave a tiny tap to her Tibetan cymbal. *Bonnng.* "Your head is an empty dome of light. Feel the space. You're here. No plans. You're now." She got to her feet. "Light seeps through your scalp and trickles down your face. Your cheeks are soft. Your mouth. Your

shoulders melt. Your arms. I'll call you back."

She moved around the room pressing down on people's shoulders. She had a brief, odd feeling of being in multiple bodies, leaning over each separate customer at the same time. And then her wristwatch drew her back. She had twenty minutes to get from here to Telegraph Hill to try to rent that perfect room.

She rang the gong and saw the customers out. The last one was Sueli, a lonely wrinkled lady who liked to talk. Sueli was the only one in the class as dark-skinned as Amy. Amy enjoyed her; she seemed like a fairy godmother.

"How many chakras do you say there are?" asked Sueli. Clearly she had some theory of her own in mind. She was very well-spoken.

"Seven," said Amy, putting on her sweats. "Why not?" She imagined she might look like Sueli when she was old.

"The Hindus say seven, and the Buddhists say nine," said Sueli, leaning close. "But I know the real answer. I learned it years ago in Sri Lanka. This is the last of your classes I'll be able to come to, so I'm going to share the secret with you."

"Yes?" This sounded interesting. Amy turned out the lights, locked the door, and stepped outside with Sueli. The autumn sky was a luminous California blue. The bay breeze vibrated the sun-bleached cardboard election signs on the lampposts—San Francisco was in the throes of a wide-open mayoral election.

"Some of us have millions of chakras," continued Sueli in her quiet tone. "One for each branch of time. Opening the chakras opens the doors to your other selves."

"You can do that?" asked Amy.

"You have the power, too," said Sueli. "I saw it in class. For an instant there were seven of you. Yes, indeed."

"And you—you have selves in different worlds?"

"I come and go. There's not so many of me left. I'm here because I was drawn to you. I have a gift." Sueli removed a leather thong from around her neck. Dangling from the strand was a brilliant crystal. The late afternoon sunlight bounced off it, fracturing into jagged rays. The sparkling flashes were like sand in Amy's eyes.

"Only let the sun hit it when you want to split," said Sueli, quickly putting the rawhide strand over Amy's head and tucking the crystal under her sweatshirt. "Good luck."

Sueli gave her a hug and a peck on the cheek as the bus pulled up.

Amy hopped aboard. When she looked back to wave at her, the old woman was gone.

The room was three blocks off Columbus Avenue with a private entrance and a view of both bridges. It was everything Amy had hoped. But the rent was ten times higher than she'd understood. In her eagerness, she'd read one less zero than was on the number in the paper. She felt like such a dope. Covering her embarrassment, she asked the owner if she could have a moment alone.

"Make yourself at home," said the heavyset Italian lady. "Drink it in." She was under the mistaken impression that Amy was wealthy. "I like your looks, miss. If you're ready to sign, I got the papers downstairs in the kitchen. I know the market's slow, but I'm not dropping the price. First, last, and one month's damage deposit. You said on the phone the rent's no problem?"

"That's what I said," murmured Amy.

Alone in the airy room, she wandered over to the long window, fiddling with the amulet around her neck. The low, hot sun reached out to the crystal. Shattered rays flew about the room, settling here and here and here.

Nine brown-skinned women smiled at each other. Amy was all of them at the same time. Her overlapping minds saw through each pair of eyes.

"We'll get separate jobs and share the rent," said one of her mouths. "And when we come back to the room we'll merge together," said another. "We'll work in parallel worlds, but we'll deposit our checks and pay the rent in just this one."

"Great," said Amy, not quite sure this was real. As she tucked away the crystal, her nine bodies folded into one.

Walking down the stairs to sign the papers, her mind was racing. Just now she'd split into nine—but Sueli had said that, with the crystal, she could split into a million.

Out the window she glimpsed another election poster—and the big thought hit her.

With a million votes, she could be the next mayor of San Francisco.

CHAPTER TWO

Our Rich World

THERE ARE TWO SALIENT DIFFERENCES between personal computer programs and the general functioning of the physical world. One: physics happens in parallel, that is, physics is being computed everywhere at once rather than within the circuitry of a single processor chip. And two: physics seems to be analog rather than digital, that is, rather than being measured in discrete bits, physical quantities are real numbers with who-knows-how-many decimal places trailing off. In this chapter we discuss how to come to terms with and even take advantage of these realities.

This chapter's six sections are as follows:

- *2.1: Rough and Smooth.* The idealized equations of mathematical physics are continuous, but the real world and our simulations are discrete, using particle system or cellular automaton computations of finite precision.
- *2.2: Everywhere at Once.* We examine the parallel architecture of the computations embodied in classical physics and learn how they can be modeled by cellular automata.
- *2.3: Chaos in a Bouncing Ball.* By discussing the motion of a bouncing ball we see that a seemingly random physical process can, in fact, be a deterministic computation.
- *2.4: The Meaning of Gnarl.* The most beautiful naturally occurring forms and motions correspond to class four computations.

- *2.5: What Is Reality?* How do we keep quantum mechanics from spoiling everything for universal automatism? Must we accept a lack of determinism at the very smallest scales?
- *2.6: How Robots Get High.* A computational view of quantum mechanics leads to the new field of quantum computation—and suggests some odd views of the mind.

2.1: Rough and Smooth

On the one hand, some aspects of the world are discrete or rough—like a pile of rocks. These rough magnitudes are best measured by counting. On the other hand, things like a person's height seem smooth or continuous and are measured in terms of size intervals. Rough things change in abrupt jumps, while smooth things ooze.

Mathematicians codify the distinction by speaking of the integers and the real numbers. Integers are fairly easy to understand: 0, 1, 2, 3, and so on. But the so-called real numbers are rather fictional: a real number between zero and one is to have a form like 0.12378909872340980... with a supposedly endless series of digits stretching out to the right of the decimal place. Ever since the nineteenth century, this infinite extravagance has been believed to be the best way to model the intuitive notion of a continuous interval as being endlessly smooth. But it may be that the real numbers of mathematics aren't a true reflection of the actual world.

Computer scientists model real numbers by finite patterns of bits, with the number of bits being something that can be adjusted for the particular application. The commonly used data types known as *float, double,* and *long double* correspond, for instance, to decimal numbers with, respectively, seven, fifteen, and thirty digits. The real numbers of computer science are "granular" in the sense that they are incremented in small steps of a minimum size.[26]

In physics we have a kind of granularity as well. Although nobody's sure what happens at the very smallest scales, quantum mechanics seems to tell us that it doesn't make sense to speak of ordinary space at scales less then what's known as the Planck length. This length is 1.6×10^{-35} meters, expressible as a decimal number whose first nonzero digit appears in the thirty-fifth place.

Planck length ~ 0.00000000000000000000000000000000016 meters.

If it doesn't make sense to speak of measuring any physical length to a greater precision than the Planck length, this means that physical coordinate locations can really have at most thirty-five digits of precision to the right of the decimal. This falls a long way short of the infinite precision enjoyed by the mathematical real numbers!

Now, it's conceivable that there might be an infinitely smooth fundamental reality underlying quantum mechanics. But it's equally conceivable that ultimate reality is fully discrete and digital—indeed, this view is becoming fairly popular among physicists, who dream of turning spacetime into something like a snap-together network of nodes and links made up of quantum loops or superstrings.

Closely related to the discrete-continuous distinction is the digital-analog divide. Adding numbers with pencil and paper seems like a digital computation, while adding numbers by measuring out lengths seems like an analog computation. Counting grains of sand seems digital, and spinning a wheel of fortune seems analog—although note that we digitize the rims of our gambling wheels into distinct bands.

The general sense is that our desktop computers carry out digital computations, while real-world physics is running analog computations. A digital computer's states resemble distinct integers, while an analog computer's states are like densely bunched decimal numbers.

A virtue of digital computations is that they're relatively impervious to outside influences. To change a digital value you actually have to make a substantial change on the order of flipping a bit. Personal computer (PC) hardware designers can build in routines to detect and correct the accidental bit-flips that occur when, for instance, a cosmic ray happens to zap a chip. An analog quantity, on the other hand, can drift away from its setting by just a tiny amount. And, as we'll be discussing in section *2.3: Chaos in a Bouncing Ball,* in chaotic analog systems very small physical differences can rapidly amplify into visible effects. This said, many analog computations are robust and insensitive to noise. This is achieved by having the system incorporate some kind of averaging process.

A side effect of a computation's being digital is that it's easy to regard it as

evolving in discrete time steps, with each step changing a state value. We think of digital systems as updating themselves in ticks like a clock, and it's meaningful to ask about a computation's "next state." In an analog system, on the other hand, it's less clear-cut when a state value has changed. Suppose, for instance, that the state involves a real number expressed by, say, thirty digits. Over a period of time, the number may go from one value to another, but if the change happens in a series of small and rapid steps, we may be unable to perceive the individual steps. An analog computation may seem to slide through a series of all-but-indistinguishable states. In formulating the laws of analog computations, rather than talking about a "next state" it's more common to simply observe the system at some successive intervals of time.

The distinction between digital and analog computations isn't sharp. Consider the following attempt at a definition.

- *Definition.* A computational system is said to be *digital* if its states range over a small set of discrete possibilities, and is said to be *analog* if it has a very large number of possible states.

The point of the definition is that whether you call a given computation digital or analog depends upon your operative notions of "small" and "large" numbers of states. In practice we can speak of *any* naturally occuring computation as being more or less coarsely digital, with the very fine digital computations shading into the ones we call analog.

We think of our digital PCs, for instance, as containing patterns of zero-or-one bits, sharp and crisp as black-and-white mosaics. Analog physical systems, on the other hand, seem to have states that are like shades of gray. The shakiness of the distinction rests on the fact that, seen at a distance, a digital pattern of black-and-white tiles will appear to be an analog shade of gray. Conversely, a seemingly smooth analog gray scale is likely to be a ladder of distinct steps.

The digital-analog distinction is further blurred by the fact that a computation may be based on lower levels that may lie elsewhere on the digital-analog spectrum. Adding numbers in your head is, for instance, digital, but the electrical and chemical potentials in the neural synapses range across so many state possibilities as to be essentially analog.

In practice, we use analog to mean *fairly* precise but not *infinitely* precise.

We imagine a very large range of analog values, but not an *endless* range. Our analog computations employ, if you will, a deflated infinite precision.

When we use analog in this limited sense, the distinction between digital and analog becomes less important, for analog computations can emulate the digital, and digital computations can emulate the analog. Let's say a bit about the two directions of emulation.

Analog emulates digital. A digital PC is based on the physical analog computations of its wires and chips—putting it differently, the analog electronic system is emulating a digital PC. How can it be possible for a machine made of analog components to have digital behavior? The answer is that a computer chip uses a whole pack of electrons to store a bit of information. Yes, the chips are small, with etched "wires" on the order of a millionth of a meter across. But electrons are in some sense a billion times smaller than that. The slightest current involves a torrent of electrons; a bit is stored by a charge of perhaps a half million electrons.

At a more primitive level, the Babbage and Scheutz machines were digital computers based on the analog motions of quite ordinary objects: gears and rods and cams. Along these lines, in the 1970s, just for fun, Danny Hillis, Brian Silverman, and some other MIT computer science students built a tic-tac-toe-playing computer out of Tinkertoys.

Figure 28: Ed Hardebeck and Brian Silverman Building a Tinkertoy Computer

Note the "programming mallet" on the floor behind Brian.

Digital emulates analog. If we use enough real-number variables of sufficiently high precision, we can always emulate any analog system of limited size. Yes, science-fictionally, speaking, we might imagine some infinitely precise analog computation that can't be emulated digitally. But so far as we know, the existence of the Planck length cutoff suggests that this isn't ever going to happen. And, as I mentioned before, it may even be that at the deepest level the world is digital.

A standard computer simulation of a continuous-valued CA approximates the cells' allegedly real-number values by discrete rounded-off numbers. Some computer scientists are leery of continuous-valued CAs because they fear that this rounding-off process creates unpredictable errors that will accumulate and become amplified into large-scale irregularities of behavior. But in actual practice, all of the continuous-valued CA simulations discussed in this book have an *averaging* step—which blocks the amplification of error. That is, in the continuous-valued CAs that I discuss, a cell's new value is based on a formula involving the *average* of the neighboring cells' values. And this averaging process damps down any troublesome round-off errors.

Even so, some doubting Thomases question whether the use of, say, the four billion possible real values allowed by thirty-two-bit real numbers produces behavior that's really the same as what you'd see with an infinite range of truly continuous real numbers. I've carried out some experiments in which I have a granularity control to select how many discrete values are used to approximate real numbers. And what I've found is that once we drop below a not very high granularity level, the behaviors of the simulation don't change, at least not over the admittedly limited times that I've watched my simulations.[27]

We can view physics in three ways, and each way allows both for a digital and an analog interpretation.

- In the *mathematical physics* view, physics is like a axiomatic system in which we derive results from equations. Although all the current equations for physics are based on continuous-valued real numbers, it may be that a future, more digital, physics can formulate the world's laws in terms of discrete numbers.

- In the *particle system* view, the world's "processors" are distinct objects—either human-scale objects like tables, chairs, and balls, or primitive objects like atoms. Whether we view these processors as carrying out digital or analog computations depends on the situation. At the high level, these distance measurements appear analog, but at a very low level they may appear digital.
- In the *continuous-valued cellular automaton* view, we see the world as a doughy continuum that we can divide up into virtual cells, with each cell being viewed as a processor. Here again we can view the cell rules as being digital or analog computations, depending on how many states we suppose the cells to be able to have.

Let's say a bit more about these three approaches.

Traditional *mathematical physics* is about smooth matter and force fields varying according to nice algebraic laws. And mathematical tools such as calculus are formulated in terms of continuous real numbers. One virtue of analog computations is that they're easy for us to think about. For analog computations are an approximation to by now familiar mathematical processes involving infinitely continuous real numbers.

Mathematical physics can in some situations provide very good predictions about what physical systems will do. This approach worked well for Isaac Newton: He was able to compute the motions of the solar system right down to the moons of Jupiter by using his laws of motion, his universal law of gravitation, and a few observations.

But, as it happens, mathematical physics runs up rather quickly against the limitations of the axiomatic approach. Some sets of equations have solutions that don't happen to have any simple descriptions. And other, more obdurate sets of equations resist any mathematical solution at all. In practice it's not possible to find idealized mathematical solutions to most real-world physical systems. Some of Newton's predictions were in fact wrong. A fully accurate general mathematical solution of a system with even *three* bodies is in fact impossible.

Physicists at today's frontiers are interested in seeing whether mathematical physics might become fully digital.

The *particle system* method is a quite demanding computational approach to physics—demanding, that is, if you want to simulate it. Here we regard the mathematical laws of physics as expressing averages across a large number of discrete particles. If we say that a Scuba tank's pressure is proportional to its temperature, for instance, we're talking about the averaged-out qualities of immense numbers of individual particles. The pressure has to do with the summed force of the air molecules hammering on the tank's walls, and the tank's temperature is derived from the average speed of its molecules. (More precisely, both quantities are proportional to the *square* of the average speed of the gas molecules.) If we had enough computational power, we could simply see these laws as emerging from the statistical behavior of a humongous particle system.

The nice thing about mathematical physics, of course, is that the laws and equations that emerge from the particles systems are often fairly simple. And the field of statistical mechanics shows how and why the laws emerge. We gain clarity by viewing physics as an analog continuum instead of a frantic dogpile of atoms.

Although the positions and velocities of the particles would seem to be continuous, there is the possibility that space and time are quantized, so that particle system models would be fully digital as well, although the "particles" of such a fully digital theory might be something as primitive as loops in superstrings.

Continuous-valued cellular automata are examples of what engineers call *finite element methods.*
The idea is to divide space up into a grid and to track something like the mass or temperature or average velocity for each cell of the grid. When we use, for instance, a continuous-valued CA to model the flow of heat, each cell holds a temperature value; in the case of water waves, the cells track height above sea level.

By making the grid coarser or finer, you can trade off between the accuracy and the speed of the simulation—if the grid were as fine as the size of individual electrons, a continuous-valued cellular automaton method would be similar to a particle system simulation. But in practice, the cells are much larger than

that and, in effect, each cell is holding values based upon an average of many particles.

Continuous-valued CA methods introduce two kinds of digitization: first of all they break space and time into discrete steps, and second, they use computer-style digital approximations to the continuous quantities being simulated.

In an ultimate universal automatist dream, we might hope to find some very simple underlying CA that doesn't even have to use continuous values—Fredkin, for instance, seemed at one time to think the world could be modeled as a two-state CA. But there are serious problems with this approach, and in fact any simple digital computation of reality would probably have to be a somewhat different architecture than that of a cellular automaton. I'll say a bit more about this in section *2.5: What Is Reality?*

2.2: Everywhere at Once

One can regard our world as a huge parallel computation that's been running for billions of years.

To get a good image of physical parallelism, imagine sitting at the edge of a swimming pool, stirring the water with your feet. How quickly the pool's surface is updated! If you toss a twig into the pool, the ripples spread out in a perfectly uniform circle. How do the ripples know where to go? The patterns emerge from reality's parallel computation.

I have the following architecture in mind, which I'll call the *classical physics architecture.*

- *Many processors.* The world's computation is ubiquitous, with no superprocessor in charge.
- *One shared memory.* Reality is one.
- *Locality.* Each processor has access to only its local neighborhood.
- *Homogeneity.* Each processor obeys the same rule.
- *Synchronization.* The processors run at the same speed as one another.

Architectural requirement	**Particles**	**CAs**
Many processors	Each particle acts as a processor	Each small region of space acts as a processor
One shared memory	The collective states of all the particles is the memory	Space with its various observables is the memory
Locality	Particles only interact when they touch	Each region of space interacts only with the nearest neighboring regions
Homogeneity	All particles obey the same laws of physics	Each region of space obeys the same laws of physics
Synchronization	Time runs at the same rate for each particle	Time runs at the same rate at each location

Table 2: Two Ways to View Classical Physics as a Parallel Computation

How does nature fill this bill? I mentioned in section *2.1: Rough and Smooth* that we can usefully view the world's computations as being either particle systems or as continuous-valued CAs. The five conditions can hold in either place (table 2).

Having *many processors acting on one memory* corresponds to the basic intuition that, on the one hand, physics is happening everywhere, and, on the other hand, there is a single shared reality that contains all physical processes. If you're thinking of the world as made of particles, this reality is sometimes called *state space* and contains such information as the mass, location, velocity, and acceleration of each particle. If you prefer to think of the world as made of continuous fields, the shared reality of state space specifies quantities like density and rate of flow measured at each region of space. In either case, the numbers that specify state space are most readily thought of as analog numbers.

The issue of *locality* is less obvious. In principle we can imagine parallel processors that have global access to all of the memory. A simple example of parallelism with global memory access would be a PC with two or more

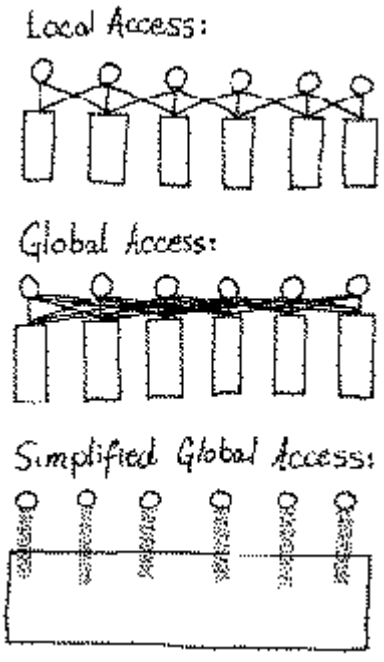


Figure 29:
Local and Global Memory Access for Parallelism

The circles stand for processors, the rectangles stand for the memory, and the lines indicate access. In the case of global access, we can simplify the figure by lumping all of the memory slots together into a single memory box and drawing a fat gray line to indicate access throughout the memory region in question.

central processing units—in the old days this required having several microprocessor chips, but now a single chip is likely to have multiple processing cores. In any case, each of the processors can access all of the PC's memory. In figure 29 we illustrate parallelism with local vs. global access.

Classical (that is, nonquantum) physics is treated as a parallel computation with *local* processor access to memory. That is, the processes at one location are affected only by the data in the immediately neighboring regions of space and time. What happens at one spot doesn't affect things somewhere else without an intervening process—such as a photon or a gravity wave. Information must be passed along in a kind of bucket brigade from one region to the next. The classical principle of locality is summarized in the slogan, "No action at a distance."

In quantum mechanics, it at first appears that locality may be violated. When two quantum systems interact and become "entangled," they can later affect each other at arbitrarily great distances, seemingly with no intervening time. In section *2.4: What Is Reality?,* I'll suggest a way in which even here some form of locality can be preserved.

The *homogeneity* condition lies at the very heart of how we imagine physics to work. There are to be certain universal laws that apply at every spacetime location. In practice, physical laws often have cases that apply only when certain extreme conditions are encountered, but the whole thrust of science is

to try to squeeze all of the conditions into one law. Extremely dense systems are the current paradigmatic example of areas where the homogeneous laws of physics run into trouble—the issue is one of reconciling quantum mechanics with the laws of gravity.

The *synchronization* condition stipulates that the processors carry out their computations at exactly the same rate, essentially updating in unison. Although this sounds like a reasonable assumption about the world's computational nature, there are serious problems with it.

First of all, Einstein's special theory of relativity tells us that if particles are moving relative to one another, then their internal clocks will in fact run at different rates. This in turn implies that any notion of "now" that we extend across a large region of space must be somewhat arbitrary. One way out might be for us to pick one privileged reference object—why not Earth—and to then adjust the rules of physics to include time dilation factors for particles moving relative to the reference object. If using Earth as the standard of rest seems too medieval, we might instead adopt a universal standard of rest based upon the cosmic background radiation—you're at rest if this radiation doesn't appear to be shifted toward the blue or the red end of the spectrum by your motion.

Fine, but once we introduce *general* relativity with its warping of spacetime, we have to deal with cusps and singular points, as at the hearts of black holes. And establishing a universal standard of rest becomes exceedingly problematic. Moreover, when we extend our considerations to the cosmological shape of the whole universe, there's a possibility that time might somehow loop back on itself. In short, if our universe is sufficiently pocked, warped, or knotted, it becomes impossible to slice it into spacelike sheets of simultaneity, and global synchronization is out of the question.

This said, the objections to synchronization need not come into play if I'm only interested in modeling some local aspect of the world, which is all I'm going to be talking about most of the time.[28]

Particle systems and CAs are both good paradigms for multiprocessor computations acting on a common memory space and satisfying parallelism, homogeneity, and synchronization. For purposes of discussion, let's consider three simple systems that we might view either as particle systems or as CAs.

- *Heat.* You dip a spoon into a hot cup of tea, and feel the warmth move up the length of the handle. In terms of particles, we might say that the molecules of the tea are moving rapidly. They collide with the molecules of the spoon and set them to vibrating more energetically. The agitated motion is passed up the length of the spoon molecule by molecule. To think of this as a CA, regard the spoon's handle as a one-dimensional row of cells. Each cell averages its temperature value with the temperatures of the neighboring cells. Repeating the averaging process over and over moves higher temperature values up the length of the spoon.
- *Water waves.* You toss a twig into a swimming pool and observe the ripples. In the particle view, pushing down the particles in one location on the surface tugs at the neighboring particles on the surface—this is the phenomenon known as surface tension. The neighboring particles in turn pull on the particles farther away, with the whole system acting something like a lot of little balls connected by springs. To view the water surface as a CA, think of the two-dimensional surface as a grid of little cells, and use the cells to model the behavior of an elastic sheet. Each cell's height above the bottom of the pool is described by the so-called wave equation, in which the rate change of a cell's height is proportional to the difference between the average height of the neighboring cells and the cell's previous height.
- *Smoke tendrils.* Someone smokes a cigarette and you watch the smoke in the air. In a particle system model, we'd say that the smoke and air molecules bounce against one another. The fine lines in the smoke pattern are visible flow lines made up of particles having the same average velocity. To see this as a CA process, think of space as made of little volume elements: three-dimensional cells. The laws of hydrodynamics relate the pressure, density, and flow direction in each cell to the corresponding quantities in the neighboring cells.

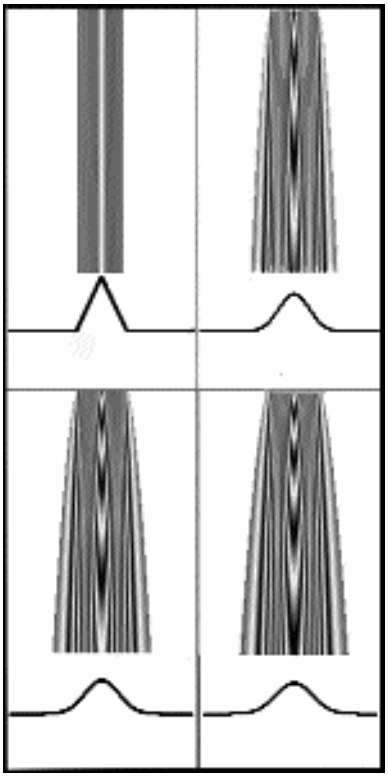


Figure 30:
One-Dimensional Heat CAs with Varying Rates of Diffusion

The figure shows four one-dimensional CA models of a wire that was heated in the middle, each wire being initially heated in the same way. The wood-grain-like pattern in the top part of each picture shows a vertical space-time sequence of successive states of the wire, with the heat values represented by shades of gray and with time running down the page. The bumpy line at the bottom part of each picture is a different representation of the heat distribution, this representation corresponding to the final instant of time. The diffusion rates a for these CAs are, left to right and top to bottom, zero, one-third, two-thirds, and one. Note that in the CA with diffusion rate zero, the heat pattern doesn't change at all, and in the CAs with lower diffusion rates, the pattern changes less than it does in the CA with diffusion rate one.

I'd like to get into some detail about how we set up CAs to model heat flow and wave motion. In describing CA rules, I like to write C to stand for a cell's current value, using New$C$ and Old$C$ to stand for the cell's next and previous values, respectively. In the case where the CA value is a single real number, NeighborhoodAverage will stand for the average of the values in the cell's neighborhood.

To simulate the flow of heat, we might use a rule of this form.

(Averaging rule) New$C$ = NeighborhoodAverage.

This might be, however, a bit crude, and lead to the heat spreading unrealistically fast. More typical is to pick a diffusion rate, a, between zero and one, and to use a rule of this form.

(Diffusion rule) New$C$ = $a \cdot$ NeighborhoodAverage + $(1-a) \cdot C$.

If *a* is 1, the Diffusion rule is the same as the Averaging rule, but as *a* gets smaller, the diffusion happens slower and slower. Figure 30 illustrates this.

We can also represent water waves by a CA rule. The rule works by having each cell take its current value *C*, add to this the average of its neighbors' values, and then subtract off the cell's previous value. In symbols,

(Wave rule) New*C* = *C* + Neighborhood Average – Old*C*.

It isn't particularly obvious that this simple rule will in fact re-create wave motion, but it works very nicely. Figure 31 shows two representations of this CA after being seeded with some randomly placed bumps—analogous to a handful of gravel thrown into a pond.

In section *1.8: Flickercladding,* I mentioned that Stanislaw Ulam collaborated with John von Neumann on discrete-valued CAs, and that in the 1950s, he switched his attention to continuous-valued CAs. One of the wins in looking at these simple toy models of physics is that it becomes possible to visualize alternative physical laws whose consequences might be too hard to understand simply by looking at the equations. And this is just what Ulam did; he began running CA models of nonstandard kinds of physics to see what would happen.

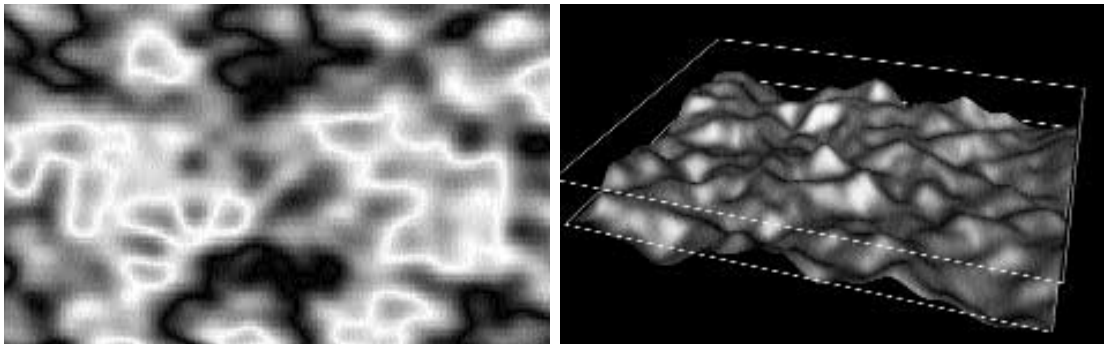


Figure 31: Two-Dimensional Wave CA, Two Views

The two views show the same CA. On the left, we represent each cell's value by a shade of gray, creating a pattern that looks like light on water. On the right, we represent each cell's value as a height, creating a three-dimensional picture.[29]

Ulam was encouraged by the atomic physicist Enrico Fermi, inventor of the neutrino. Fermi was curious about what might happen if one looked at so-called nonlinear waves. What is the meaning of "nonlinear" in this context? Ordinary waves—like the ones we just discussed simulating—are often based on a kind of spring force. If you stretch a string, or a water surface, it wants to pull back to its original size. In ordinary physics, this restoring force of a spring is proportional to the displacement—one has a Hooke's Law–type equation of the form $F = k \bullet \text{displacement}$. This is called a linear equation because there aren't any exponents in it. Fermi wondered what a wave might look like if it was acting on a substance in which the restoring force satisfied an equation with an exponent, such as $F = k \bullet \text{displacement}^2$, or even $F = k \bullet \text{displacement}^3$. As Ulam put it:

> Fermi expressed often a belief that future fundamental theories in physics may involve nonlinear operators and equations, and that it would be useful to attempt practice in the mathematics needed for the understanding of nonlinear systems. The plan was then to start with the possibly simplest such physical model and to study the results of the calculation of its long-time behavior.[30]

Working with the Fermi and the early computer scientist John Pasta, Ulam carried out the experiments and wrote them up. Figure 32 shows what the Fermi-Pasta-Ulam quadratic and cubic waves look like.

It's interesting that a mathematician of Ulam's caliber was thrown back on carrying out a cellular automaton simulation. If he wanted to know the effects of a nonlinear wave equation, why couldn't he just work out the math? After all, rather than viewing the world as a particle system or as a CA, we can also regard the world as a set of equations. So why didn't Ulam simply *deduce* what a nonlinear wave would do?

Well, nonlinear waves are an example of a situation that resists analysis by an axiomatic approach. If you want to figure out what a nonlinear system is going to do, you actually need to run some kind of simulation of it.

Or, of course, you can just look at the world itself. Ultimately, we don't really *need* to model the world. It does a fine job of computing itself. Indeed, for a universal automatist, *physics is made of computations.*

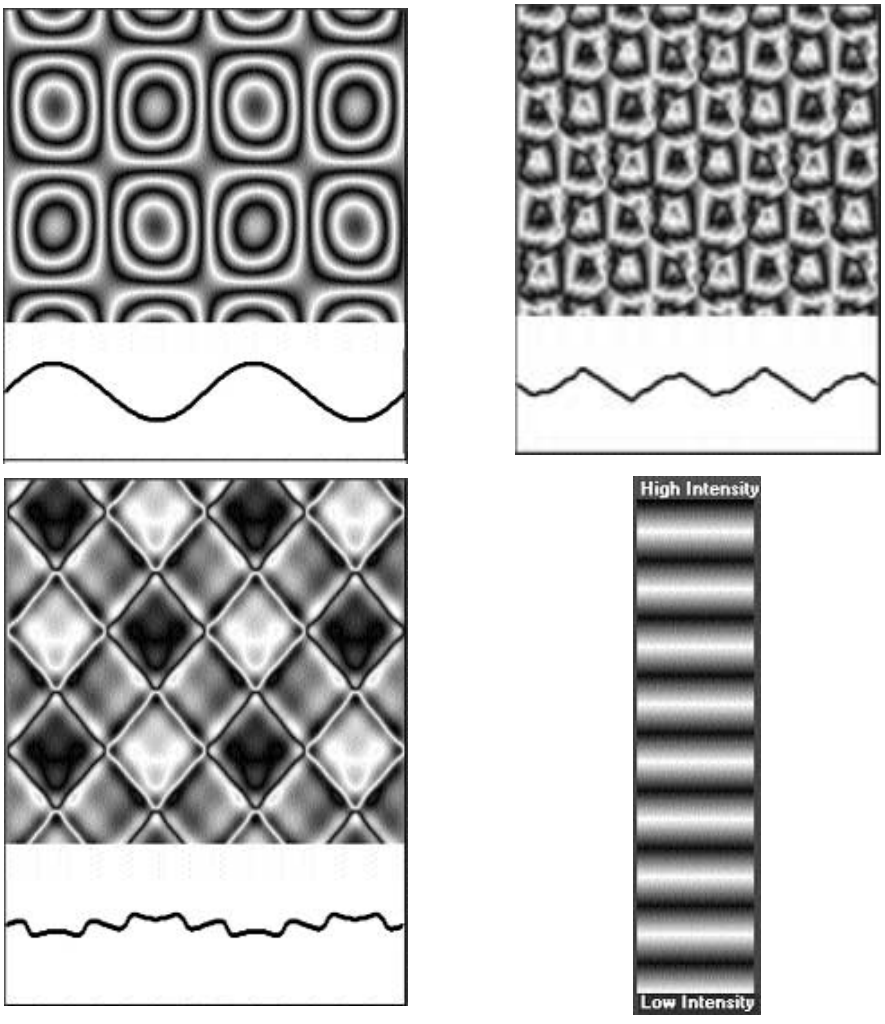


Figure 32:
Fermi-Pasta-Ulam Quadratic and Cubic Nonlinear Waves

The figure on the lower right shows the shadings that we use to indicate the cell values. The other three figures show one-dimensional CAs. In each of these figures, the upper part is a spacetime diagram, that is, a stack of a few hundred pictures of the CA, with time running down the page. Think of the alternating dark and light bands as being lines of equal altitude above the plane of the page. The wiggly graphs at the bottom are instantaneous pictures of the CA state, with the cell values represented as vertical positions; here the altitude direction is within the page. In each case the CA is seeded with a smooth wave at start-up. The upper left image shows a quadratic wave after a hundred updates, when it still looks very much like a regular wave. The upper right image shows the same quadratic wave after about fifty thousand updates. The lower left shows a cubic wave after about fifty thousand updates.[31]

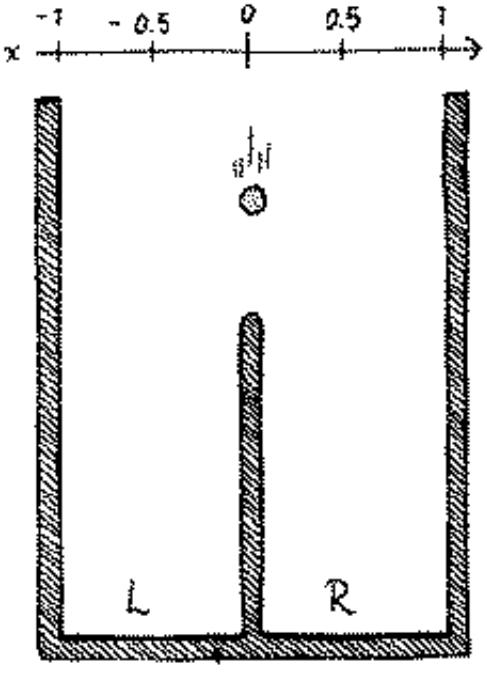
2.3: Chaos in a Bouncing Ball

In this section, we'll revert to the particle system view and regard the world's processing elements as being ordinary objects; in particular, I'll talk about the motions of balls. My first example is what I'll call the bin experiment (figure 33).

A ball drops from a fixed height straight down to a box divided into two tall bins, and the ball ends up in either the left or the right bin, with no possibility of bouncing from one bin into the other.

We suppose that we can vary the ball's starting position along the horizontal or "*x*"-axis, with the zero position located exactly above the center of the partition dividing the two bins. We might summarize this by saying the experiment is a Bin(*x*) process that computes a Left or Right bin output from the starting position *x*.

For the moment, we'll ignore the parallel aspects of physics and focus on the ball as a single serial processor. For the ball, the low-level software is the laws of physics, the high-level software is the configuration of the bin, the initial input is the ball's starting position *x*, and we're interested in the output state where the ball has settled down into the left or the right bin.



Remembering the stored program concept, we recognize that there's not that sharp a boundary between the high-level software and the input data. In other words, you can either think of the box as "software" or "data." I'm leaning toward the software view, as I'm thinking of situations where we might throw a ball into some fairly complicated and mazelike collection of passages and walls and ask about where the maze design makes the given input ball end up.

Figure 33:
A Ball Computes Bin(*x*) to Be Left or Right

Although the simple bin experiment seems nice and deterministic, there are some quagmires to explore.

When the ball is released near the center of the *x*-axis, it may bounce up and down on the divider for a while. Maybe once in a blue moon it'll end up balanced upon the divider for a longer period of time. Is this a problem? Not really. We can just say that there are *three* possible outputs: at a given time the output Bin(*x*) might be Left, Right, or Up—where the Up state corresponds to any situation where the ball hasn't yet landed and come to rest.

If we're going to talk about time, we might as well make it explicit, and write Bin(*x, t*), to indicate the state of the system *t* seconds after we drop the ball from position *x*. Bin(*x, t*) will be Up for the smaller values of *t*; then eventually it will go to Left or Right and stay there. And if we just write Bin(*x*), we mean, let's say, the value of B(*x*, 300), that is, the state of the ball five minutes (or three hundred seconds) after you drop it.

Fine, but now we have to face a harder problem. Suppose you were to actually set up a bin experiment and carry out a large number of runs, each time dropping the ball from what seemed to you to be the exact center of the *x*-axis. The resulting outputs would be a more or less random sequence of Left and Right outputs, with maybe, once every billion runs, an Up output that lasts a full five minutes. But rare, anomalous cases aren't the important issue. The important issue is that if we keep dropping the ball from what seems to be the exact center, our bin experiment will generate a seemingly random sequence of results. Although we think we're using the same input over and over, we keep getting different results.

The best way to express this is to say that an individual physical computation like dropping a ball into the bin is *not repeatable.* We can *approximately* repeat many physical computations—otherwise we'd never learn to walk or throw a Frisbee. But the scuzz and fuzz of the natural world keeps its computations from being *precisely* repeatable.

The reason is that we do not—and cannot—have perfect control over the *x* input. The value of *x* may be *near* zero, but it won't be exactly zero. And each time we put it near zero, it'll be "near" in some different way. You can't set *x* to the same value twice in a row because it doesn't make sense to say that a normal-size object's location is *exactly* some precise number.

Suppose, for instance, you're measuring the position *x* in meters, and

you're trying to get a lot of decimal places of accuracy. When you reach the ninth decimal place, you're at the nanometer level, where the sizes of molecules and atoms become significant. At this scale, your ball is a vibrating cloud of atoms, and asking about the exact center of the cloud becomes as impractical as asking after the exact center of a swarm of gnats.

The more decimals you want, the worse it gets. At eighteen decimals, you're down at the level of protons and neutrons, and most of the ball looks like empty space. And at the thirty-fifth decimal place you hit that troublesome Planck length, the scale at which continuous space may not even exist. Measure a position to an arbitrary precision? Forget it!

Can't our uncertainty about the ball position's smaller decimal places just stay insignificant? No. Suppose that the divider between the two bins has a rounded top. Geometrical considerations show that each bounce moves the ball farther from the exact center. The amplification is in fact exponential, in the sense that after n bounces, an initial displacement will be on the order of 10^n times as big. Another way to put it is that each bounce brings another few decimal places of the position into visibility. No matter how tiny the initial displacement is, it will rather quickly become visible (figure 34).

To complicate things, as the ball bounces on the divider, effects from the irregularities in the surfaces of the ball and of the divider come to dominate the not-really-so-precise initial condition. Before long, you have to consider the effects of air currents and even the gravitational effects of objects other than the downward-pulling earth. These influences are, if you will, interactive inputs added onto the initial input. If the ball bounces long enough on the divider, no effect is too small to have an influence.

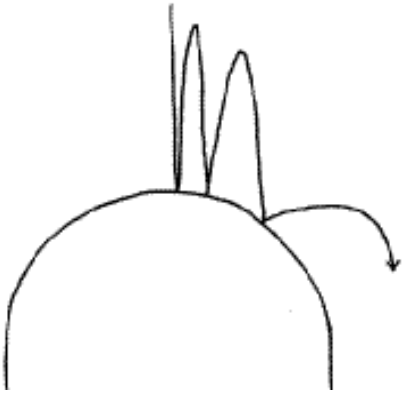


Figure 34: Bounce Magnification

So what are we left with? Is the bin experiment a deterministic process? Can we call it a computation?

Yes. It's a computation that just doesn't happen to be *repeatable*—that is, you can never manage to reset things to get the

exact same initial input and then observe the exact same series of outputs. The bin experiment is unrepeatable because the dynamics of this system amplifies the tiniest details of the initial and interactive inputs so that they have large and noticeable effects.

This point is important, so I'll say it once more. When we get into the zone near the center point, the bin experiment remains deterministic, but it becomes sensitive not only to the details of the input position but also to vagrant influences by the environment. Yes, the output seems random, but this is only because the initial and interactive inputs aren't fully known. And it's this lack of knowledge that makes the experiments unrepeatable.

The bin experiment is an example what physicists call a *chaotic* system. A chaotic system is one that rapidly amplifies the details of its initial conditions and external influences. In the formal definition of chaos, mathematicians also require that a chaotic system is one that will now and then appear to be periodic for short periods of time. Very many, perhaps most, everyday physical systems are chaotic.

Note that although chaos makes processes complex, that doesn't mean that these computations are random. Waves and clouds are chaotically diverse, but they do tend to have certain characteristic patterns. When you go to the beach, you don't see a completely random goulash of water and air—no matter how gnarly the surf, it's still made up of waves in a characteristic distribution of sizes.

The characteristic space and time patterns of chaotic processes are known as *strange attractors.* The science writer James Gleick describes how a group of Santa Cruz chaoticians known as the Dynamical Systems Collective learned to see them in the natural world.

> They had a game they would play, sitting at a coffeehouse. They would ask: How far away is the nearest strange attractor? Was it that rattling automobile fender? That flag snapping erratically in a steady breeze? A fluttering leaf?[32]

Drop a piece of paper and watch it drift to the floor. The paper seesaws back and forth, twirls, flips over, dives, and catches itself with a sudden swoop. And if you drop it again it's likely to do something different. Repeatedly

toss an apple core toward a trash can. Now and then you may seem to be in a groove, with the core bouncing in, but over time, the results are quite unpredictable. Observe a drop of milk spreading through your coffee. There is a certain regularity to the tendrils, but nothing long-lasting or systematic. Run your fingernail across your desk and listen to the sound. Make your bed and regard the exact shape of the crease where the blankets tuck in. Watch a raindrop on a windowpane. All of these systems are rule-based and deterministic. Yet all of them continually produce surprise. These and perhaps most other physical systems are computations that are in practice unrepeatable because you can never reproduce the exact same combination of initial and interactive inputs.

Some chaotic systems explode into a grungy thrashing, while others settle into very nearly repetitive behavior patterns. Chaotic systems can range from having a lesser or a greater amount of disorder.

A key distinction between bouncing balls and PCs is that our PC computations *are* repeatable. This is because PCs are digital, with a feasibly small range of initial values and because they are well isolated from unwanted inputs.

But because bouncing balls are analog, their computations are not repeatable. The difference between analog systems and digital systems is not that the analog computations are in any way less accurate. The difference is that analog systems have so many states that it's physically impossible to control the inputs of an analog computation precisely enough as to make it repeatable.

As it happens, the physical computations we enjoy watching are the least likely to be repeatable. In a ball game we relish the moments when the ball's motion is the most obviously chaotic. The football that dances and flubbed on the players' fingertips. The basketball that circles the hoop before dropping in. The line drive that escapes the pitcher's glove to be bobbled by the short stop and caught by the second baseman. The Ping-Pong shot that skids along the net and somehow crawls over it.

Back in the 1970s, my family and I lived in upstate New York. There was a relentlessly lively little boy in our neighborhood. Kenny. We had a wooden garage with rafters, bookcases, bicycles and tricycles, sleds on the walls, rakes and hoes, a lawn mower, a rabbit hutch, and so on. Kenny would throw a discarded tennis ball into our garage as hard as he could, and excitedly describe the paths the ball would take. "Look, it hit the paint can and slid off the hose

onto the windowsill and rolled across to the bicycle seat before it dribbled under the car!" Kenny was having fun watching physical computations.

Observing physical computations is a simple human pleasure. Last night, in fact, with these ideas in my mind, I was playing at the dinner table. My wife and I were guests of the neighbors, and I picked up a plastic wine-cork and tossed it toward the wooden napkin ring lying on its side on my host's place mat. The cork landed just right, did a gentle flip, slid into the napkin ring, and stayed there. Goal! I spent the next few minutes trying to do it again, until my wife made me stop. At social gatherings, a gentleman eschews fanatical computer hacking of any kind.

Physical computations are things we can enjoy with our whole bodies. One of the particular joys of mountain biking is riding down a hill, enjoying the sensations of a computation playing itself out. The hill is the input, physics is the low-level software, your bicycle and your reactions are the high-level software, and the output is your breezy ride.

Bicycling, or for that matter skiing, involves a downhill ride that's chaotic in its sensitivity to small influences. After a bit of practice, you learn to supply a stream of interactive inputs that guide you away from outputs involving a spill. Physiologists report that a human brain sends out muscle control signals at a rate of about ten pulses per second. Using the rapid computer in your head, you're able to predict the next few seconds of your onrushing physical computation and to tailor your control pulses to guide you toward the outputs you prefer.

When an ongoing computation adjusts itself—like a bicyclist, a skier, or, for that matter, a soaring bird—we see intelligence. But the motions of even dumb, unguided physical objects can be arbitrarily complex.

Recall Wolfram's Principle of Computational Equivalence (PCE), which I introduced in section *1.2: A New Kind of Science.* Wolfram claims that essentially all complex computations are universal, that is, rich enough to emulate any other computation. In other words, most of the physical systems we encounter are universal, just as they are.

What kinds of examples do I have in mind when I speak of universal physical computations? Chaotic ones. A ball bouncing around a cluttered garage. Ice crystals forming in freezing water. Drifting masses of clouds. Amplified feedback from an electric guitar. A scarf dropping to the floor.

How simple can a universal physical computer be? In the 1970s, the autodidact universal automatist Edward Fredkin described how to make a universal computer from billiard balls bouncing around on a frictionless table. It's not much of a leap from this to arguing that the three-dimensional motions of the air's molecules are also a universal computer.

But do keep in mind that in the cruddy, scuzzy world of physical objects, the motions of air molecules or Fredkin billiard balls are irredeemably chaotic, rapidly amplifying the slight inaccuracies of the starting conditions and then being swamped by external effects like the tiny gravitational forces from the motions of the observer. The bouncing particles will compute *something,* but probably not what you intended them to.

I mentioned in section *2.1: Rough and Smooth* that we can digitally emulate physics, at least in principle. But in practice there are three difficulties: a first relates to *initial conditions,* a second comes from the *supplemental inputs* we just mentioned, and a third concerns *unfeasibility.*

Suppose your targeted task is to emulate precisely some particular run of a physical process. You plan to make, say, a virtual model of my garage, and toss in a virtual ball and try to get it bounce around just like Kenny's tennis ball did.

The *initial condition* problem is as follows. Because each bounce amplifies more digits into visibility, you have an exceedingly low probability of exactly emulating Kenny. Yes, as you hunt through the range of possible inputs you may come across runs that start out by behaving like Kenny's big throw, but the simulations will eventually diverge from the reality as you simulate more and more of the elapsed time.

The *supplemental inputs* problem has to do with the fact that even if you miraculously match Kenny's initial input, due to the chaotic sensitivity to supplemental inputs, even if a trajectory matches Kenny's for a second or two, it will soon veer away as a result of tiny supplemental inputs from the world at large.

The *feasibility* problem has to do with the fact that even our most highly parallel digital computers have a minuscule number of computational nodes compared to nature. What analog systems lack in repeatability they gain in their massively parallel powers of computation.

Yes, you can simulate a ball bouncing around a garage quite well because here you're essentially ignoring the parallelism of the physical world. But now suppose I ask you to also simulate the air in the garage. It's hopeless to try to individually simulate each of the astronomical number of atoms, and even if you go to a higher-level finite-element model, it's impossible. Think of all the eddies that form in the wake of the ball, not to mention the vibrations from Kenny's shrill voice—the waves and vortices crossing one another and bouncing off the irregular objects, fleeting flows interacting in gnarly, nonlinear ways.

Practically speaking, digital computers have no hope of feasibly emulating the full richness of the physical world in real time. But we can be consoled by the fact that we already *have* the world, and it's already a computation.

The dream of traditional physics is to find simple laws to describe nature. In some cases, such as the motions of the planets in their orbits around our sun, simple formulas can go a very long way. But when you get into detailed, chaotic situations like a ball bouncing around a cluttered garage, you often need to fall back upon detailed simulation—which still doesn't give very good predictions, as it's impossible to specify the initial conditions to a high-enough degree of accuracy.

Some physicists dislike Wolfram's work because he brings them bad news. Recall Wolfram's PCU.

- *Principle of Computational Unpredictability (PCU).* Most naturally occurring complex computations are unpredictable.

The PCU tells us that most physical systems are going to be unpredictable in the formal sense that there isn't going to be a simple and rapidly running computation that can emulate the physics very much faster than it happens.

A crucial point that I'll be returning to is that the unpredictability is *not* just the result of sensitive dependence on initial conditions and on supplemental inputs. Even if an experiment could be conducted in an utterly shielded environment with a strictly accurate initial condition, the computation itself would be unpredictable.

One of Wolfram's favorite examples along these lines involves the motion of

a projectile, such as a cannonball. In an idealized experiment where you ignore the effects of air friction, if you fire a bullet into the air, the bullet's *velocity* in feet per second and *height* in feet at a given *time* will be given by *simple equations* of this form.

$$\text{velocity} = \text{startvelocity} - 32 \cdot \text{time}$$
$$\text{height} = \text{startheight} + \text{startvelocity} \cdot \text{time} - 16 \cdot \text{time}^2.$$

The beauty of these equations is that we can plug in larger values of time and get the corresponding velocity and the height with very little computation.

Contrast this to *simulating* the motion of a bullet one step at a time by using a rule under which we initialize velocity to startvelocity and height to startheight and then iterate the following two *update rules* over and over for some fixed time-per-simulation-step *dt.*

Add $(-32 \cdot dt)$ to velocity.
Add $(\text{velocity} \cdot dt)$ to height.

If your targeted *time* value is 10.0 and your time step *dt* is 0.000001, then using the simple equations means evaluating *two* formulas. But if you use the update rules, you have to evaluate *two million* formulas!

The bad news that Wolfram brings for physics is that in any physically realistic situation, our exact formulas fail, and we're forced to use step-by-step simulations. Real natural phenomena are messy class three or gnarly class four computations, either one of which is, by the PCU, unpredictable. And, again, the unpredictability stems not so much from the chaoticity of the system as it does from the fact that the computation itself generates seemingly random results.

In the case of a real object moving through the air, if we want to get full accuracy in describing the object's motions, we need to take into account the flow of air over it. But, at least at certain velocities, flowing fluids are known to produce patterns very much like those of a continuous-valued class four cellular automaton—think of the bumps and ripples that move back and forth along the lip of a waterfall. So a real object's motion will at times be carrying out a class four computation, so, in a formal sense, the object's motion will be unpredictable—meaning that no simple formula can give full accuracy.

This summer I passed a few hours in the Museum of the History of Science in Geneva, Switzerland. It's a jewel of a place, a dozen small rooms in the two stories of a lakeside mansion, with parquet floors and enchanting prospects from every window, and most of the windows open to catch the breezes from the lake. Glass cases hold brass scientific instruments: microscopes, telescopes, barometers, Leyden jars, spectroscopes, and the like. It stands to reason that these precision instruments would be found here in the nation of watchmakers; indeed, quite a few of them are Swiss-made.

In the museum, I photographed what seems a perfect image for science's dream of finding a simple explanation for everything: the crank on an orrery. An orrery is a tabletop model of the solar system, you see, with a little handle that you turn to move the planets and moons in their orbits.

How *about* the minuet of the planets, spinning to the stately music of Newton's laws? It's well known that Newton's laws can be used to formally derive the simple laws of planetary motion known as Kepler's laws. Does this mean that the solar system's motions are fully predictable?

No, even here chaos and unpredictability raise their untidy heads. In 1987, the computer scientists Gerald Sussman and Jack Wisdom carried out a monster simulation of the solar system to show that, in the long run, the motion of Pluto is chaotic.[33] The formal derivation of Kepler's laws doesn't take into account the pulls of the planets upon one another, and once we include this in the mix, Kepler's music of the spheres becomes discordant. We get, once again, a class-four computation, unpredictable by any means other than a detailed simulation. Deterministic yes, predictable no.

Figure 35: The Secret Machinery of the Universe

The little crank you turn to move the planets of an orrery in the Geneva Museum of the History of Science. These days, we instead use computer programs that are so-called digital orreries.

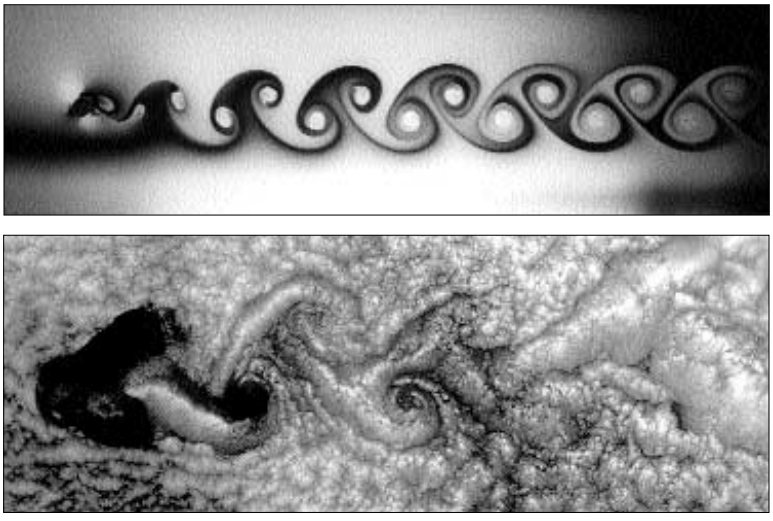


Figure 36: Von Karman Vortex Streets

The top image was created by Maarten Rutgers using a flowing soap film, and the bottom is a satellite photo of the clouds near the island of Guadalupe.

Wolfram feels that the successes of science are limited to a small number of phenomena, and that in most situations we will have to fall back on the computationally intensive process of simulating the evolution of the systems about which we want to make predictions. Indeed, it seems likely that most natural processes can't be predicted in detail by any simple formula—if for no reason than that there are so many processes, and so few simple formulae, that there aren't enough "elegant laws of nature" to go around![34]

2.4: The Meaning of Gnarl

Building computations in layers is a recurrent theme—think of a computer game powered by the microcode of a chip, or of a human reckoner whose thoughts are the firings of neurons. Layers upon layers of computation, emulations upon emulations.

One of the nicest words I've picked up from my philosopher friends is *phenomenology*. Phenomenology is the study of what you actually experience—

independent of the theories and explanations that you've been taught. Phenomenologically speaking, continuous classical physics is closer to reality than stories about atoms. There's no need to apologize or feel inauthentic if you take an observant layman's view of the physical world. If you see it, it's real.

If you start looking around for computation-like physical processes, you'll find them everywhere. Some of the most dramatic examples occur in fluid flow. Figure 36 shows a particular fluid-flow phenomenon called *von Karman vortex streets* after the Hungarian aeronauticist Theodor von Kármán. When a stream of air or water flows around an obstacle at moderate speed, eddies appear in the wake. Examples would be the vortices you see trailing your hand when you sweep it through the water of a sunlit pool, the whirlpools that seem to spawn off the back of a rock in a stream, the exquisitely filigreed smoke from a steaming cup of tea, or the great whorls that form in cloud formations downwind from mountains.

As the situation gets more complicated, so do the patterns. Figure 37 shows a photo I took of a stream flowing around four rocks at the Esalen Institute (naturally) in Big Sur. The bumps in the water surface were fairly

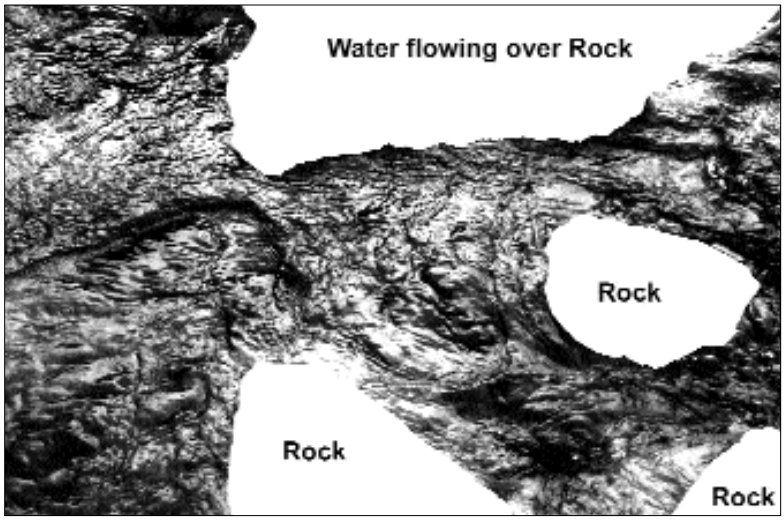


Figure 37: Water in a Stream

Looking down with the water flowing from left to right. Rocks are whited out for clarity.

stable, almost like solid objects, although now and then twisting and shifting in response to slight changes in the amount of water coming through.

To some extent these gnarly patterns result from the intricate initial conditions of the streambed and the supplemental inputs of the downstream flow. But there's another factor to keep in mind, the self-generated structures of the flow computation itself. The stability of the patterns suggests that the patterns aren't *wholly* determined by the inputs. The chaotic computation has strange attractors that it tends to settle in on.

If you had a totally smooth streambed, what kind of patterns might you see on the water's surface? You might suspect that very simple and uninteresting ripple patterns would result. But this seems not to be the case. Any fluid flow is rich enough to generate random-looking patterns quite on its own. You don't need any supplemental inputs to churn things up. The computation is class four on its own.

Stephen Wolfram remarks that we might take a high-level view of flowing fluids, treating, say, vortices as objects in their own right. Rather than saying the complex motions of the vortices are the result of chaotically amplified inputs, it might be possible to explain the motions in terms of a fairly simple computational rule about the vortices themselves. It may be that many of the external disturbances are averaged away and damped down—and the gnarly patterns we see are the result of a simple high-level computation that happens to be unpredictable.

Recall here that I say a computation *P* is *unpredictable* when there is no shortcut computation *Q* which computes the same results as *P*, but very much faster. Wolfram also speaks of such computations as *irreducible* or as *intrinsically random.*

I might mention in passing that computer scientists also use the word *pseudorandom* to refer to unpredictable processes. Any programming environment will have built into it some predefined algorithms that produce reasonable random-looking sequences of numbers—these algorithms are often called pseudorandomizers. The "pseudo" refers to the fact that these are in fact deterministic computations.[35]

Recall that Wolfram's Principle of Computational Unpredictability (PCU) says that most naturally occurring complex (that is, not class one or class two) computations are unpredictable. Putting the PCU a bit differently, we

expect to find that most naturally occurring complex computations are intrinsically random or, if you prefer, pseudorandom.

To illustrate the notion of intrinsic randomness, Wolfram points out how the evolution of a one-dimensional cellular automaton can produce unpredictable patterns, starting from an initial condition of a few marked cells and with no supplemental inputs at all. As I'll discuss in just a minute, Wolfram's favorite example of an intrinsically random process is the two-valued CA Rule 30. But first, for a change of pace, look at the intrinsically random continuous-valued CA in figure 38.

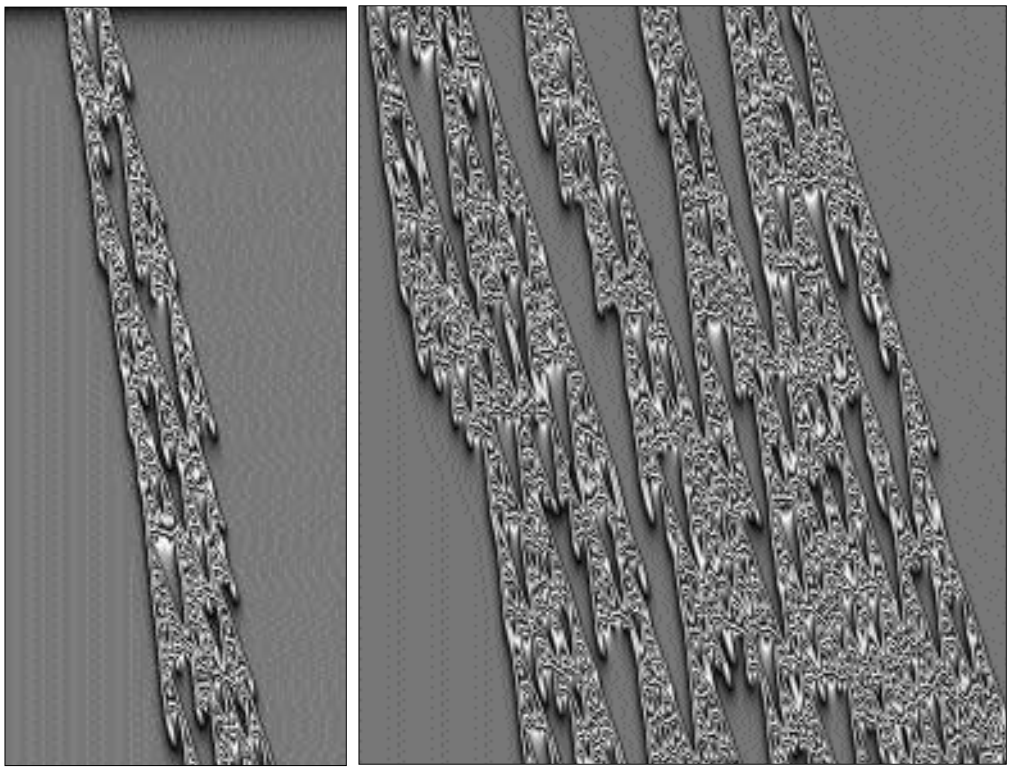


Figure 38: A Deterministic One-Dimensional CA Creating Gnarly-Looking Flow

The left side shows the first six hundred generations of a line seeded with a smooth bump of values in its center. The right side shows generations 8,000 to 8,600. The rule is called Honey, and is a continuous-valued rule based on taking weighted averages of cells with their neighbors, using different averaging methods according to whether the cell's value is positive or negative.

As is customary in a CA, the cells are updated in parallel, which means that during each full update of the cellular automaton, every cell on the tape computes a new value for itself. The way to read figure 38 is to view space as the horizontal axis and time as running down the page. What we see are successive copies of the cellular automaton's tape. Each row of black, white, and gray cells represents one successive step of the computation. The picture is, if you will, a spacetime diagram.

The idea behind intrinsic randomness is that not all of the world's seeming randomness needs to result from outside agitation and the chaotic amplification of initial conditions. Some, or perhaps most, of nature's complexity can arise from intrinsic randomness—from a simple computation endlessly munching on the same region of data and pumping out unpredictable new patterns.

Now you might think that the intrinsic randomness of the Honey rule has something to do with its use of continuous-valued real numbers. Maybe it's excavating hidden initial conditions out of the real numbers with which I seeded it. This is why Wolfram's favorite poster child for intrinsic randomness is so important—there's absolutely nothing up the sleeves of the one-dimensional CA Rule 30.

Recall that in Rule 30, the cell values consist of a single zero-or-one bit, which we represent, respectively, by white or black. If we start Rule 30 with *one single black cell,* it quickly fills up the right half of the tape with a class-three pattern resembling the foam in a beer glass (see figure 39).

If you repeat the run, you get exactly the same pattern, so it's deterministic. Yet anyone looking at the sea of triangular bubbles in the bottom right half of the picture would imagine the system to be random. The moral is that a deterministic world is perfectly capable of generating its own randomness. This is unpredictability; this is intrinsic randomness.

Recall that Wolfram's Principle of Computational Equivalence (PCE) proposes that most naturally occurring class three and class four computations are equally complex. But observationally, there's a distinction between the two.

Although class-three computations are intriguing, the most beautiful computations are class four. These are the ones that I call "gnarly."

The original meaning of "gnarl" was simply "a knot in the wood of a tree." In California surfer slang, "gnarly" came to be used to describe complicated,

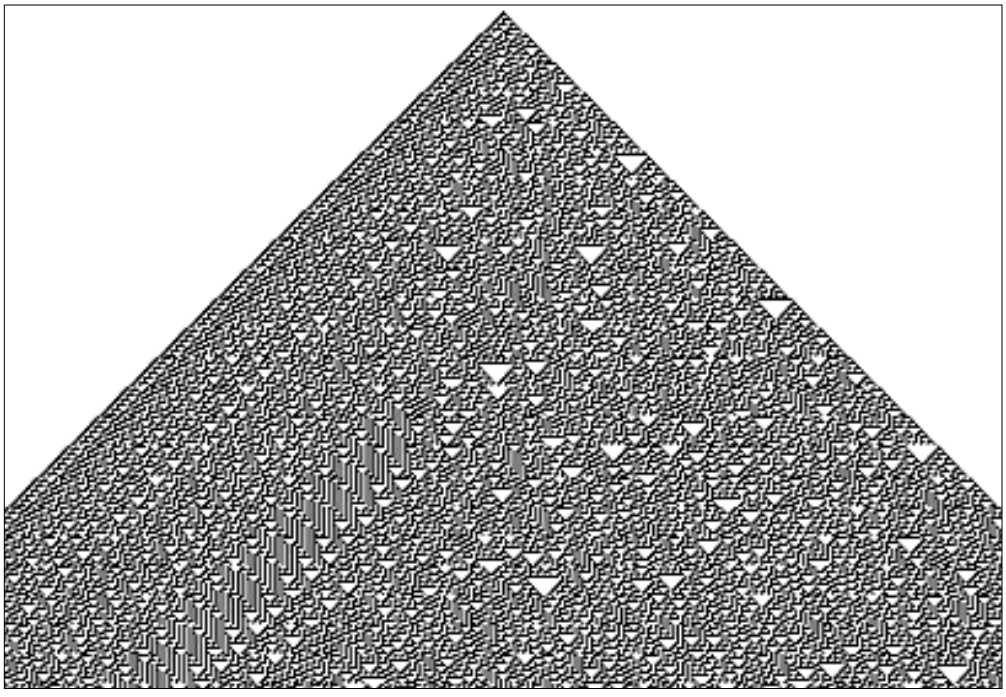


Figure 39: Rule 30, Started from a Single Black Cell

rapidly changing surf conditions. And then, by extension, something gnarly came to be anything with surprisingly intricate detail.

Do note that "gnarly" can also mean "disgusting." Soon after I moved to California in 1986, I was at an art festival where a caterer was roasting a huge whole pig on a spit above a gas-fired grill the size of a car. Two teenage boys walked by and looked silently at the pig. Finally one of them observed, "Gnarly, dude." In the same vein, my son has been heard to say, "Never ever eat anything gnarly." And having your body become old and gnarled isn't necessarily a pleasant thing. But here I only want to talk about gnarl in a good kind of way.

Clouds, fire, and water are gnarly in the sense of being beautifully intricate, with purposeful-looking but not quite comprehensible patterns (figure 40). And of course all living things are gnarly, in that they inevitably do things that are much more complex than one might have expected. The shapes of tree branches are the standard example of gnarl. The life cycle of a jellyfish

Figure 40: "As Above, So Below"—Gnarly Clouds and Water

is way gnarly. The wild three-dimensional paths that a hummingbird sweeps out are kind of gnarly, too, and, if the truth be told, your ears are gnarly as well.

Let's come back to the surf at an ocean beach. As I already mentioned in the previous section, *2.3: Chaos in a Bouncing Ball,* although the patterns of the water are clearly very complicated, they aren't random. The forms of the waves are, from moment to moment, predictable by the laws of fluid motion. Waves don't just pop in and out of existence. Water moves according to well-understood physical laws. It's a deterministic computation.

You might notice that the waves near a rock tend every so often to fall into a certain kind of surge pattern. This recurrent surge pattern would be a chaotic attractor. In the same way, chaotic computer simulations will occasionally tighten in on characteristic rhythms and clusters that act as chaotic attractors. In either case, we're dealing with a class four computation.

If there is a storm, the waves may become quite choppy and disorderly. This is more like a class three computation. As disorderliness is increased, a chaotic physical system can range from being nearly periodic, up through the visually interesting region of the strange attractors, and then into uniform seething. This, again, corresponds to the passage from class two to class four to class three computations. As I mentioned before, Wolfram's number-ordering for his computational classes is a bit misleading. Class four is in some sense *between* classes two and three, as I suggested in figure 7.

The reason people might think waves are random is because the computation that the water performs is many orders of magnitude larger than

anything our computers can simulate. Yes, simulating actual waves on an electronic computer is unfeasible. But that doesn't mean that waves aren't the result of deterministic computations being carried out by the physical world. And we have every reason to suspect that these computations are class four and unpredictable.

The great discovery we've made with our personal computers is that you don't need a system as complicated as the ocean to generate unpredictable gnarl. A very simple rule can produce output that looks, at least superficially, as complicated as physical chaos. Unpredictable computer simulations are often produced either by running one algorithm many times (as with the famous Mandelbrot set) or by setting up an arena in which multiple instances of a single algorithm can interact (as in CAs).

We find the same spectrum of disorder across a wide range of systems—mathematical, physical, chemical, biological, sociological, and economic. In each domain, at the ordered end we have class one constancy and a complete lack of surprise. One step up from that is periodic class two behavior in which the same sequence repeats itself over and over again—as in the structure of a crystal. Adding a bit more disorder leads us into the class four or gnarly zone, the region in which we see interesting behaviors. And at the high end of the spectrum is the all-but-featureless randomness of class three.

Regarding physical matter, in classical (prequantum) physics, a vacuum is the simplest, most orderly kind of matter: nothing is going on. A crystalline solid is orderly in a predictable, periodic way. And fluids such as liquids or gasses are fairly disorderly, more along the lines of being class three. Matter is computationally at its most interesting when it's near a phase transition, as when a liquid is freezing or coming to a boil. Matter near a phase transition to some extent has a nested class two structure, with similar kinds of features occurring at widely different scales. But the phase transition structure is very dynamic, with information-laden patterns moving about, and is, I believe, best thought of as class four.

The flow of water is a rich source of examples of degrees of disorder. The most orderly state of water is, of course, for it to be standing still. If one lets water run rather slowly down a channel, the water moves smoothly, with perhaps a regular class two pattern of ripples in it. As more water is put into a channel, eddies and whirlpools appear—turbulence. If a massive amount of water is

Level of Disorderliness	Lower	High	Higher	Highest
Subregion of the gnarly zone	Quasiperiodic	Strange attractors	Chaotic bifurcations	Pseudorandom

Table 3: Subspectrum of Disorderliness for the Gnarly Zone

poured down a steep channel, smaller and smaller eddies cascade off the larger ones, ultimately leading to an essentially random state in which the water is seething. The gnarly zone is where the flow has begun to break up into eddies with a few smaller eddies, without yet having turned into random churning.

In every case, the gnarly zone is to be found at the interface between order and disorder. In the mathematics of chaos theory, we can refine this a bit more, distinguishing four subregions of the gnarly zone (see table 3).

The most orderly kind of gnarly behavior is quasiperiodic, or nearly periodic. Something like this might be a periodic function that has a slight, unpredictable drift. Next comes the strange attractor zone in which the system generates easily visible structures—like the gliders in a CA rule, or like standing waves in a stream. Then we enter a critical transition zone, which is the heart of the gnarl.

In the language of chaos theory, a system undergoes a *bifurcation* when a system switches to a new attractor. This is when a system begins ranging over a completely different zone of possibilities within the space of all possible phenomena. The term *bifurcation* is a bit misleading, as a chaotic bifurcation doesn't necessarily have anything to do with something splitting into two. Bifurcation means nothing more than changing something about a system in such a way as to make its behavior move to a different attractor.[36]

As we turn up the disorder of a gnarly system, the system begins experiencing bifurcations in which one strange attractor repeatedly gives way to another. Initially the system may be dancing around on, say, an ellipse, and a moment later, the successive points may be scattered about on something shaped like a bow tie.

And at the highest end of disorder we shade into the pseudorandom chaotic systems, whose output is empirically indistinguishable from true randomness—unless you happen to be told the intrinsically random algorithm that is generating the chaos.

My favorite example of gnarly physical chaos is a tree whose branches are gently trembling in the breeze (figure 41). Here's some journal notes I wrote about gnarl and a tree that I saw while backpacking in the Los Padres Wilderness near Big Sur with my daughter Isabel and her friend Gus in May 2003.

> Green hills, wonderfully curved, the gnarly oaks, fractal white cloud puffs, the Pacific Ocean hanging anomalously high in the sky, fog-quilted.
>
> I got up first, right before sunrise, and I was looking at a medium-sized pine tree just down the ridge from my tent. Gentle dawn breezes were playing over the tree, and every single one of its needles was quivering, oscillating through its own characteristic range of frequencies, and the needle clumps and branches were rocking as well, working their way around their own particular phase space attractors, the whole motion harmonious in the extreme. Insects buzzed about the tree, and, having looked in the microscope so much of late, I could easily visualize the micro-organisms upon the needles, in the beads of sap, beneath the bark, in the insects' guts—the tree a microcosmos. The sun came rolling up over the ridge, gilding my pine. With all its needles aflutter it was like an anemone, like a dancer, like a cartoon character with a halo of alertness rays.
>
> "I love you," I said to the tree, for just that moment not even needing to reach past the tree to imagine the divinity behind it, for just that moment seeing the tree as the body of God. "I love you."
>
> When we got home there were my usual daily problems to confront and I felt uptight. And now, writing these notes, I ask how can I get some serenity?
>
> I have the laptop here on a cafe table under a spring-green tree in sunny blue-sky Los Gatos. I look up at the tree overhead, a linden with very small pale fresh green leaves. And yes the leaves are doing the hand jive. The branches rocking. The very image of my wandering thoughts, eternally revisiting the same topics. It's good.
>
> The trees, the leaves, the clouds, my mind, it's all the same, all so beautifully gnarly.

Figure 41:
Tree, Cloud, Mind, Mountain

2.5: What Is Reality?

Years ago I was discussing far-out science-fiction ideas with the beloved mathematics writer Martin Gardner. I was describing the notion that if you could measure a totally rigid piece of material to endless precision, then the successive digits might code up, say, the true and complete story of your life. What if everyone were born clutching a personal talisman of this kind in his or her hand—like program notes summarizing the action of an opera? And suppose that, instead of going on and living a real life, some poor guy wastes all his time decoding his talisman—only to learn that he'll spend his entire allotted span measuring one little object!

But, thanks to atomism, we can't really measure much past twenty digits, and even if we could, quantum mechanics makes space fairly meaningless out past the thirty-fifth digit. So the idea doesn't quite work.

"Too bad," said Martin. "Quantum mechanics ruins everything."

Physicists have great confidence in quantum mechanics because it predicts, among other things, the precise values of certain physical constants.

But physical constants have to do with quantities that are measured as averages over many different runs. When it comes to predicting how any individual particle will behave, quantum mechanics is usually helpless.

An example. Suppose we have a beamsplitter (see figure 42), that is, a partially reflecting mirror that reflects half of all incoming light and transmits the rest. And let's say that we have a pair of light-sensitive detectors labeled 0 and 1. The transmitted light goes to 1 and the bounced light goes to 0. The 0 and 1 detectors each register an intensity half as great as that of the incoming beam. But what happens if we send in a beam that contains only a single photon?

One of the brute facts about nature is that light is quantized; that is, you actually *can* turn down a light beam's intensity to a specific minimal intensity that sends one photon at a time. Photons are supposedly indivisible—you can't get half a photon arriving at 0 and half a photon arriving at 1. So the single-photon beamsplitter system has to make what seems to be a random choice. If you repeat the experiment over and over, about half the photons end up at 0 and about half end up at 1.

We had a similar result in our bin experiment, and in that case we were satisfied with saying that there were tiny differences in the initial conditions and external influences on each successive ball. Why can't we take a similar approach with photons encountering a beamsplitter? The photons could be like balls, the beamsplitter could be like the bin divider, and the various outcomes could deterministically depend on tiny details.

But physicists say this is impossible. They insist that we *can't* explain the variation by saying the photons might be hitting the mirror in slightly different locations, that the photons might be slightly different, or that there are external influences nudging the photons this way and that. They say there's no hope of finding a deeper explanation of the photons' decisions, and that an individual photon's choice of whether to trigger detector 0 or detector 1 is fundamentally and inexplicably random.

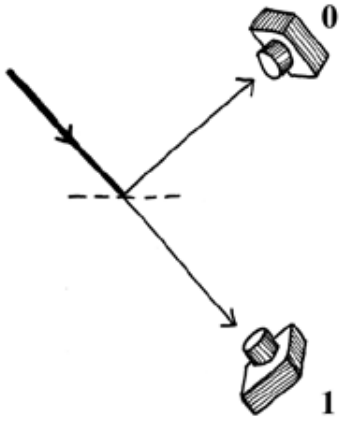


Figure 42: The Beamsplitter

In the words of the endlessly quotable physicist Richard Feynman, with his own characteristic italics:

> Yes! physics *has* given up. *We do not know how to predict what would happen in a given circumstance,* and we believe now that it is impossible, that the only thing that can be predicted is the probability of different events. It must be recognized that this is a retrenchment in our earlier ideal of understanding nature. It may be a backward step, but no one has seen a way to avoid it. . . . We suspect very strongly that it is something that will be with us forever—that it is impossible to beat that puzzle—that this is the way nature really is.[37]

Feynman is always persuasive, but physical determinism is not so dead an option as he suggests. My opinion is that there are in fact underlying psychological reasons driving the conventional insistence that we should welcome quantum mechanics and the destruction of determinism. Often when I hear a popular lecture on quantum mechanics, I detect a lilting, mystery-mongering tone. "Be happy! The universe is incomprehensible! How wonderful!"

The rejection of determinism seems to provide some people with a sense of liberation. The hidden part of the argument might go like this: If the world is fundamentally random, then surely I'm not a robotic machine, and if I'm not a machine, then perhaps I have an immortal soul, so death isn't so frightening.

Now that I've delivered this ad hominem attack on the advocates of quantum mechanics, I must, in all fairness, admit that I have my own psychological reasons for not wanting to view quantum mechanics as a final answer. First of all, like many mathematicians, I'm uncomfortable with uncertainty. In this vein, it could also be that a lifetime's worth of hard knocks has taught me that when there are no rules, most people get a raw deal. A second point is that I like thinking of the universe as a single entity that's at some level knowable; I like to imagine that, if you will, I can see the face of God. And this dreamed-of cosmic unity becomes less plausible if the universe results from an all-but-infinite collection of utterly random bit-flips with absolutely no common underlying cause.

As I say, I think the mystifications of quantum mechanics seem appealing precisely because people would like there to be some escape from the logical and deterministic fact that we're all going to die. But if the fear of death is indeed the issue, why not find solace in thinking of the universe as an immense logical system of which you're a tiny part? When your particular pattern ceases to exist, the grand computation will continue. Your alloted region of spacetime will "always" be around. Can't that be enough? For that matter, what's so terrible about death? Personally, I don't really *mind* the notion of someday getting off the stage and no longer having to continue my long-winded computation—but maybe that's just because I'm getting old.

Let me make one more point. If you fear that determinism means you're a machine without a soul, consider that, given what we know about class-four computations, there's no reason to think that we can't be both deterministic *and* unpredictable, no reason to think that your soul couldn't in some sense *be* a gnarly computation. Consider: The world could be perfectly deterministic and still look and feel exactly the same as it looks right now. Indeed, I think that's the true state of things. Quantum mechanics simply doesn't go deep enough. And we have nothing to lose by moving beyond it to a fully deterministic universal automatism.

Enough rhetoric; let's get back to science. There seem to be two kinds of reasons why physicists don't expect photons to behave like balls.

The first reason is that photons are meant to be elementary particles, without any of the nicks and dings that can serve to explain why balls act unpredictably. Well—maybe so, maybe not. It's at least conceivable that photons themselves are the averaged-out results of still more fundamental phenomena—not necessarily subparticles, but possibly something like network patterns or linked loops in a multidimensional superspace.

The second, more compelling, reason that photons aren't like balls is that they're also like waves. The photon is in some sense a wave that takes *both* paths through the beamsplitter, and the presence of the detectors makes the smeared-out wave collapse into a single photon at 0 or a single photon at 1. And—here's that same bad news again—the outcome of any individual photon wave collapse is to be completely random.

This odd sequence of spreading-wave-followed-by-collapse-into-particle is a standard pattern in quantum mechanics. Any system is to be thought of as

an abstract wave that obeys deterministic analog laws until some kind of measurement is performed on the system. And the measurement process forces the spread-out wave to collapse into a single definite state. The possible outcome states depend on the kind of measurement being made, and the probabilities of the various outcomes depend on the wave.

One can carry out a fairly simple experiment to demonstrate that a single photon can indeed act like a wave that takes both paths through the beamsplitter. The idea is to arrange two beamsplitters and two regular, non-beamsplitting mirrors to make a device known as an interferometer, as shown in figure 43. A light beam coming in from the upper-left-hand side will split into the bounced 0 and the transmitted 1 beams. These in turn will uneventfully bounce off the mirrors at the top and the bottom, and when these beams strike the beamsplitter on the right, they'll split again, yielding four beams that we might as well call: 00, 01, 10, and 11. The history of these four beams' encounters with the beamsplitters and the normal mirrors can be summarized, respectively, as bounce-bounce-bounce, bounce-bounce-transmit, transmit-bounce-bounce, and transmit-bounce-transmit.

By, let us say, turning an adjustment screw, you can tweak the position of the upper mirror in the system so that beam 01 reinforces the beam 10. And you'll find that when you do this, beams 00 and 11 interfere with each other, effectively canceling each other out The effect is that all of the light coming in from the upper left seems to leave along the lower right direction. And this works even if we send in only one photon at a time.

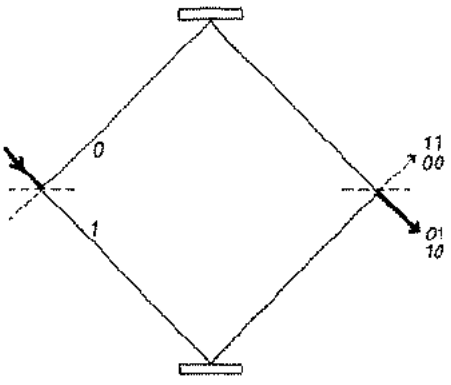


Figure 43: Interferometer

In order to understand what's going on, we think of the photon as a wave that gets split up. A pair of waves will enhance each other if they're in phase with each other, that is, if their crests match. And they'll cancel each other if one wave's crests match the troughs of the other. And bringing 01 and 10 into synch puts 00 and 11 out of synch. The reason has to do with the total number of mirror-bounces by each wave; each time a wave bounces off a mirror of any

kind, its so-called phase shifts by 90 degrees, and waves that are 180 degrees out of phase cancel each other out.

By the way, once we get the interferometer tuned like this, it's also the case that if a photon comes into the system from the lower left corner, it will end up exiting in the upper right direction; we'll return to this point in this chapter's final section, *2.6: How Robots Get High,* when we'll view an interferometer as being like a logical NOT gate.

But for now, the big deal about the interferometer experiment is that it works even if we pass *only one* photon through the system at a time. That is, if you send in a single photon from the top left, you always get a single photon exiting from the bottom right. The only conclusion seems to be that the "indivisible" photon somehow behaves like a wave that splits into four beams, two of which cancel each other and two of which reinforce each other. So a photon is a wave.

But when we go back to the first experiment of this section and just run the photon through *one* beamsplitter, we see the spaced-out wave inexplicably making nondeterministic random choices between 0 and 1.

At this point we're supposed to be so confused that we give up and agree with the quantum physicists that the world is nondeterministic and we'll never really understand it, and isn't it great to have the world be fundamentally incomprehensible and mysterious? "Come forward, dear friends, come drink the Kool-Aid."

Sigh.

It's like I'm at the beach and a kid kicks down my children's sand castle, and when I go to scold the kid, his mother says, "Oh, he's not like your children. He's special. You'll never understand how he feels. It would be quite impossible for you."[38]

Do we *really* have to let quantum mechanics kick a hole in our sand castle? A lot of it has to do with how we choose to interpret the empirical facts of quantum mechanics. The standard way of viewing quantum mechanics is called the *Copenhagen interpretation,* but there are various alternate ways to look at things. Here I'll only discuss three of the possible ways to restore determinism.

A *first* attempt is to say that as long as we're primarily interested in the behavior of the medium-size objects of daily life, we're looking at statistical

averages in any case, and any quantum mechanical randomness is averaged away. And, as we've already discussed, the large-scale smoothed-out laws of physics are quite deterministic.

But you can imagine situations where quantum mechanical effects are greatly amplified. You might, for instance, connect the 0 and 1 detectors of the beamsplitter to, say, music players in two different locations, and when a single photon goes into the beamsplitter there will be music here or music there, but not both. Or if that doesn't sound portentous enough, replace the music players with hydrogen bombs. The point is that in principle an individual quantum event can be amplified into a normal-size event, which is then happening nondeterministically. Our lives can in fact be affected by quantum events in a less contrived way. Consider the fact that it takes but one unfortunately placed radon atom's decay to lethally mutate a gene in a newly fertilized egg.

There's an even more serious problem with any plan to dismiss quantum mechanics and act as if the world is really classical: It's thanks only to quantum mechanics that our atoms and molecules are stable. If an electron could have any orbit at all around a nucleus, it would quickly spiral inward and the atom would collapse. Nature uses the kinky strictures of quantum mechanics to make the electron keep a proper distance. Like it or not, quantum mechanics is an integral part of daily life.

This said, the actual cases where quantum indeterminacies become visible are surely quite rare. And nothing would really look any different if it turned out that these seemingly random quantum events were in fact directed by an underlying class three or class four computation.

A *second* defense of physical determinism is the many universes theory, which insists that there are a vast number of parallel universes tied together into a so-called multiverse. When faced with some seemingly random quantum choice, the multiverse responds by picking both options, spawning off new universes as necessary. So then it looks as if we have regained a kind of determinism: the photon goes to both 0 and 1. Every plane crashes in some branch of the multiverse; every lottery ticket is somewhere a winner. But how does the multiverse view explain why your particular world is the way it is? The move is to claim that "you" are in lots of parallel universes. In one world you're seeing that photon go to 0 and in another you're seeing it go to 1.

Many people find the multiverse model philosophically unsatisfying. It's hard to put one's finger on the problem, but I think it has to do with *meaning*. One likes to imagine the world has an ultimate explanation of some kind. Call it what you will: the Secret of Life, God's Plan, the Theory of Everything, the Big Aha, whatever. If we live in a multiverse of many universes, then perhaps the multiverse has a Multiversal Big Aha, but a mere individual universe doesn't get a Big Aha. An individual universe is simply the result of an incalculable number of coin flips.

To me, this feels inane and defeatist. Our beautiful universe deserves a better explanation than that. Although the multiverse model is in fact useful for understanding certain kinds of quantum phenomena, it's not attractive as a final answer.

A *third* defense of determinism suggests that quantum mechanics depicts particle behavior as random only because it doesn't go deep enough. Quantum mechanics seems so odd precisely because it *isn't* actually a final, complete, and fundamental theory of reality.

It's well known that quantum mechanics doesn't merge well with general relativity, and physicists are exploring any number of more fundamental theories, such as string theory and loop quantum gravity.[39] While Einstein's general theory of relativity was inspired by a specific geometrical vision of curved space, quantum mechanics seems to have arisen as the haphazard result of symbol pushing and mathematical noodling. Although quantum mechanics works, it lacks a sensual core that would compel wholehearted assent. Many physicists say this is simply because the microworld is essentially different from the world in which we live. But it's not unreasonable to suspect that a radically different theory awaits us and that determinism could still be regained.

In particular, universal automatists such as Edward Fredkin and Stephen Wolfram feel that there *is* a deterministic fundamental theory based on simple computations of some kind. Fredkin has been known to argue that the world is made of cellular automata, and Wolfram takes a more sophisticated approach involving networks and systems of symbol transformations.[40] Wolfram compares present-day quantum mechanics to a theory that studies the temperatures and pressures of gases without being aware that a gas is

made up of atoms. Seemingly fundamental entities such as photons, electrons, and their wave functions may in fact be emergent patterns based upon a low-level sea of computation.

The computational view of quantum mechanics is in effect what's known as a hidden variables theory. One has to be careful with hidden variables, for theoretical and experimental work in quantum mechanics tell us that if we gain some comprehensibility by assuming the world has real and definite underlying states, then we have to pay a price by accepting weirdness of some other kind.

As an example of hidden variables, consider a situation known as the Einstein-Podolsky-Rosen paradox. Here two particles with a common past event *O* are observed to behave "synchronisitically" at some later times. That is, if particle *A* and particle *B* were at one time tightly coupled, then if you later make a measurement on particle *A*, you may get some random-seeming value, but if you happen to measure *B* as well, you'll get the same value from *B*. And this works even if the measurements on *A* and *B* are too widely separated to be able to send slower-than-light signals to each other. In this kind of situation we say that *A* and *B* are entangled.

Now a simplistic hidden-variables interpretation might suggest that the answers to the measurement were hidden in *A* and *B* all along and that they adopted a common setting at *O*. But subtle statistical experiments have ruled out this option—in some sense it seems that *A*'s state really isn't determined until it's measured, and at that point *B*'s state becomes determined as well.

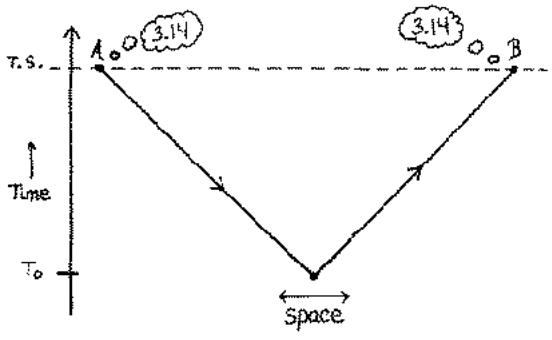


Figure 44: Reversed Time Signals Explain Synchronicity

A sends a signal backward in time to O, *which sends a signal forward in time to* B.

A more sophisticated kind of hidden-variables theory takes a spacetime view and says that the future measurement on *A* is a kind of hidden variable that reaches back in time to *O* and forward from there to *B* (see figure 44). The outcome of the measurement is, if you will, a variable that's hidden in the

future. The physicist and science-fiction writer John Cramer has worked out a worldview like this that he calls the *transactional interpretation of quantum mechanics*.[41]

Cramer proposes that we change how we *think* about quantum mechanics. But there's no real change in the *predictions* that are made. The standard Copenhagen interpretation, the multiverse interpretation, and Cramer's transactional interpretation are different mental models for the same sets of facts.

In Cramer's model we have hidden future events and we also have signals that travel backward in time. In this transactional interpretation of quantum mechanics, any event sends signals into both the future and the past. An observation of, say, a photon emission occurs via a kind of handshaking link whereby a forward signal from cause to effect is paired with a backward signal from effect to cause. By allowing time-reversed backward signals, you also can have quantum mechanical effects that jump instantaneously across great distances, as I indicated in figure 44. In Cramer's model, the entire future is fixed, with the forward and backward effects acting as threads to weave reality into a consistent whole.

Note that if all of time is linked, then there's no real point in distinguishing one particular slice as the "now." Everything fits into a whole. This lends some credence to the Jungian notion of synchronicity, which supposes that meaningful coincidences really do occur and that life indeed has the same carefully plotted quality as a novel or a myth.

In my book *The Fourth Dimension* I point out that this notion was anticipated by the fourteenth-century mystic Meister Eckhart in one of his sermons:

> A day, whether six or seven ago, or more than six thousand years ago, is just as near to the present as yesterday. Why? Because all time is contained in the present Now moment.
>
> To talk about the world as being made by God tomorrow, or yesterday, would be talking nonsense. God makes the world and all things in this present now. Time gone a thousand years ago is now as present and as near to God as this very instant.[42]

Fine. But if past-present-future are a single integral whole, the explanation (if any) of this lovely synchronistic spacetime tapestry needs to come from

somewhere else. Of course at some point, most explanations end up turning to an inexplicable prime mover (aka God), but let's see how much we can cut back on the work the prime mover needs to do.

Cramer offers no explanation of *why* we have our particular spacetime tapestry; indeed, he himself is not particularly wedded to determinism. My interest in his interpretation stems from the fact that it *does* make determinism possible. But it's determinism of an odd kind.

My idea is to combine a Wolfram-style view of reality with Cramer's transactional interpretation. Suppose with Cramer that causality runs both forward and backward in time, and also suppose that our world is deterministic in both these temporal directions. This means that spacetime is a coherent whole, with both past and future fully determined by the world's state at any single instant. If you fix upon some arbitrary moment in time—say, the instant when you read this sentence, then the question becomes: How was the world's structure at this particular instant determined? If you can explain the now, you get the entire past and future for free—for the past and future follow deterministically from the now.

Now I add in the Wolframite element. Think like a universal automatist and suppose that the great structure of quantum-mechanically patterened spacetime arises from a higher-dimensional deterministic computation. Since our time-bound human nature makes its easier to imagine a deterministic computation as being embedded in some kind of time, let's invoke a (possibly imaginary) second time dimension in which to compute our world—call this extra time dimension *paratime.* Paratime is perpendicular to our ordinary dimensions of space and time, and we want the entire universe to be the result of a computation that's taken place in the direction of paratime, as illustrated in figure 45.

Note that the paratime notion reintroduces the theme of parallel worlds. Presumably the people in each of the spacetimes feel themselves to be in a unique reality with time flowing forward as usual. Note also that, if we take the paratime view seriously, it's possible or even likely that the spacetime in which we find ourselves isn't the last one in the series. Reality evolves further along the paratime axis. In terms of an analogy to a novel, our world is very well plotted, but it may not be the final draft.

I had a momentary sensation of an flow of paratime while I was working

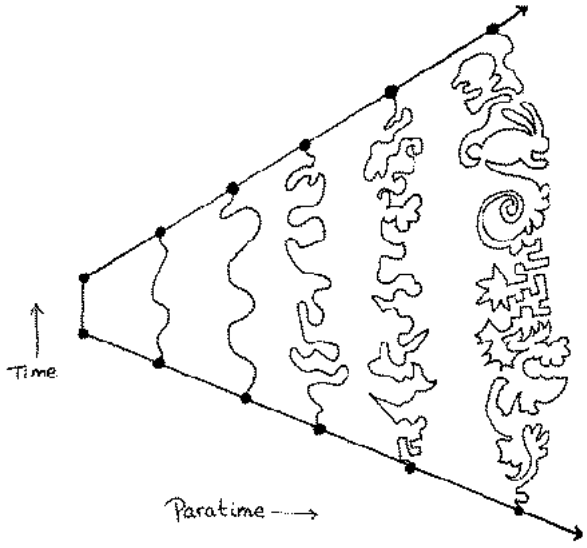


Figure 45: Evolving a Spacetime across Paratime

Think of the six lines as six increasingly evolved spacetimes, each of which runs from an initial dot to a final dot. The leftmost spacetime can be thought of as a simple seed that a computational rule transforms across paratime into a more complex spacetime.

on my historical novel *As Above So Below,* about the life of the sixteenth-century Flemish painter Peter Bruegel. In the course of detailing Bruegel's life, I was focusing my successive chapters on individual paintings by the master. Now, Bruegel's best-known series of paintings is called *The Seasons,* and consisted of six panels representing different times of the year. My researches had led me to believe that he painted them in his studio in Brussels, and that in January 1566 he transported them to a patron's house in Antwerp, using a horse-drawn cart called a Belgian wagon. While I was trying to visualize this, something strange happened, which I recorded in my writing notes.

> I'm finally writing the chapter on *The Hunters in the Snow* (figure 46). I've been a little scared of this one. It's a big chapter I've looked forward to.

Figure 46: Peter Bruegel's *Hunters in the Snow*

> Just now I had a kind of spooky-feeling experience. I figured out that Peter would be using a Belgian wagon to haul his six *Seasons* pictures up to Antwerp, and I was wondering if a wagon like that could make it through the snow, and I looked over at the *Hunters in the Snow* reproduction that I have on my wall by my desk, and it felt like there was this twinkling in the middle of the picture, and then all of a sudden there was a Belgian wagon there (figure 47).
>
> I'm imagining, just for fun, that the Belgian wagon didn't "used" to be in the *Hunters in the Snow.* That in fact Bruegel's pictures are changing a little bit as I write about them. But the changes are uniform across all of spacetime, so when my copy of *Hunters in the Snow* changes, so do all the others, and all of everyone's memories about the picture. Reality shifts to a slightly different parallel sheet. And I only notice this at the instant it happens, and even then I can never be sure.[44]

Rather than saying every possible universe exists, I'd say, rather, that there is a sequence of possible universes, akin to the drafts of a novel.

We're living in a draft version of the universe—and there is no final version. The revisions never stop.

Figure 47: Bruegel's Belgian Wagon

Detail of Hunters in the Snow.

Each draft, each spacetime, each sheet of reality is itself rigorously deterministic; there really is no underlying randomness in the world. Instead we have a great Web of synchronistic entanglements, with causes and effects flowing forward and backward through time. The start of a novel matches its ending; the past matches the future. Changing one thing changes everything. If we fully know everything about the Now moment, we know the entire past and future of our particular sheet of spacetime.

To make this discussion seem just a shade more reasonable, let's look at a CA model. Recall that the pictures of one-dimensional CAs take the form of spacetime diagrams, with the horizontal axis representing a ribbon of space and the vertical axis corresponding to time. Now it turns out to be fairly easy to construct "reversible" cellular automata for which the past and the future both follow from the present. In these physicslike CAs, no information is lost, and anything that happens in one direction of time can equally well happen in the other direction. Figure 48 shows the spacetime of a CA of this type.[43]

A reversible rule of this kind serves as a model for a transactional-quantum-mechanics world where events send out effects both forward and backward in time. We might think of the reversible CA rule as the world's physics. Everything in a reversible world like this hinges on the state of any single spacelike slice—or what we've been calling a "Now moment."

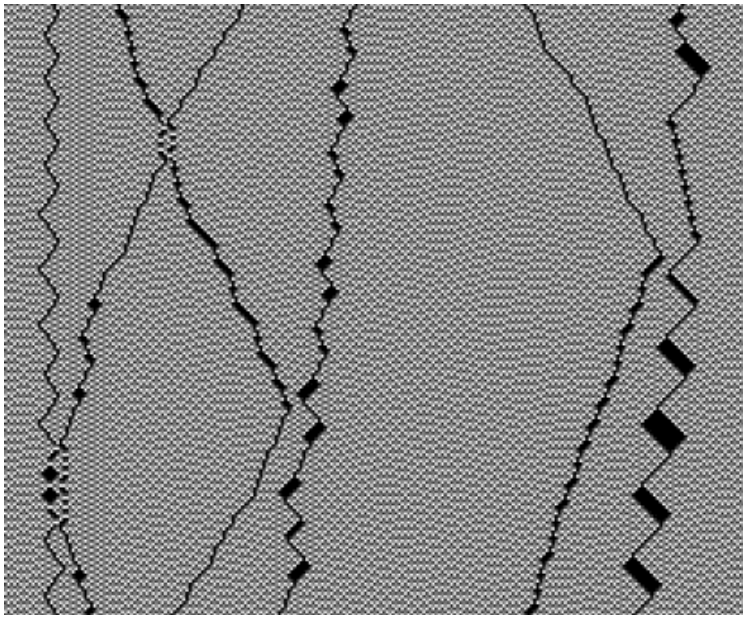


Figure 48: Spacetime Diagram of a Reversible One-Dimensional CA

This is the Axons rule of the Cellab software.

As I said above, the universal automatists would like to find a seed and rule that could compute across paratime to generate the spacetime in which you and I are living our lives. And if the physics within spacetime is deterministic towards both future and past, it would be enough to find a seed and a rule that could compute across paratime to produce one particular "now" slice of our spacetime. And then the past and the future could be deterministically generated from the now.

With this in mind, explaining a given draft of the universe becomes a matter of explaining the contents of a single Now moment of that draft. This, in turn, means that we can view the evolution of the successive drafts as an evolution of different versions of a particular Now moment. As Scarlett's climactic scene with Rhett is repeatedly rewritten, all the rest of *Gone With the Wind* changes to match.

And this evolution, too, can be deterministic. In other words, we can think of there as being two distinct deterministic rules, a Physics rule and a Metaphysics rule, as shown in figure 49.

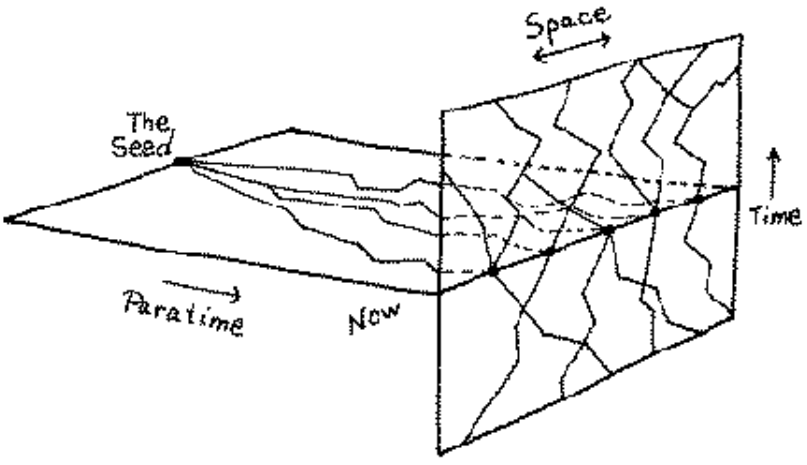


Figure 49: A Physics and a Metaphysics to Explain All of Spacetime

In this picture we think of there being two distinct CA rules, a Physics rule and a Metaphysics rule. The vertical plane represents our spacetime, and the line across its middle represents a spacelike "Now." The Physics rule consists of time-reversible laws that grow the Now moment upward and downward to fill out the entire past and future of spacetime. And we invoke the Metaphysics rule to account for the contents of the Now moment. The Metaphysics rule is deterministic but not reversible; it grows sideways across a dimension that we might call paratime, turning some simple seed into the space-filling pattern found in the Now.

The Metaphysics rule is—what? The Metaphysics rule is like a CA that grows the space pattern from some presumably simple seed. When I speak of this metaphysical growth as occurring in paratime, I need only mean that it's logically prior to the existence of our spacetime. We don't actually have to think of the growth as being something that's experientially happening—as I was suggesting with my *Hunters in the Snow* example.

The Metaphysics rule could be something as simple as an eight-bit cellular automaton rule generating complex-looking patterns out of pure computation. Or perhaps the Metaphysics rule is like the mind of a Great Author creating a novel, searching out the best word to write next, somehow peering into alternate worlds. Or, yet again, the Metaphysics rule could be the One cosmic mind, the Big Aha, the eternal secret living in the spaces between our thoughts.[45]

The message to take away is that quantum mechanics doesn't *have* to spoil everything after all. It's just a bag of tricks that some mathematical physicists

made up. Reality may very well be a deterministic computation based on rules no more intricate than the rules of cellular automata.

2.6: How Robots Get High

Whether or not quantum mechanics is a final theory of reality, the fact remains that it's a very powerful and intellectually rich system. So now let's set all doubts aside and see what we can learn from it. After all, even if quantum mechanics is in some sense incomplete, any future physics will undoubtedly incorporate quantum mechanics as an approximation—in much the same way that quantum mechanics includes classical physics as an approximation that holds for larger-sized objects.

In this section I'll discuss three topics:

- Quantum coherence as a metaphor for the human mind.
- The dream of quantum computation.
- The computational architecture of quantum mechanics.

Under the traditional Copenhagen interpretation of quantum mechanics, measuring a quantum system changes its state in an abrupt and unpredictable fashion (see table 4).

Even worse, if you want to measure two properties of a system, the answers you get will depend on the order in which you make the measurements. It's a little as if you had a picture book, and if you look at the pictures, the words in the book are no longer the same, and if you read the words, the pictures are altered. Not at all like repeatedly reading information off a disk.

The notion of quantum indeterminacy can be expressed in terms of *superposed states,* which serves, if nothing else, as a very useful metaphor for the human mind. "Superposed" connotes having multiple layers overlaid and merged.

Quantum mechanics tells us that any measurement you make on a system carries with it a set of expected answers—these are the so-called pure states or eigenstates of the measurement. When you measure a system, it enters one of the measurement's unambiguous or "pure" states. The system is effectively forced to pick one answer out of a fixed list of multiple choice options. This transition happens abruptly and discontinuously and is called the *collapse of the wave function.* The collapse of the wave

Kind of State:	Superposed	Pure
Arises:	Naturally, via Schrödinger's wave equation	After a measurement
In terms of pure states:	Sum of pure states	One pure state
Process producing state is:	Deterministic	Random
Relation to environment:	Coherent or partly decoherent	Fully decoherent

Table 4: Mixed and Pure States

function is an irreversible process; you can't restore a system to the state it was in right before you measured it.[46]

Different kinds of measurements have different sets of pure states. As I discuss in the long footnote 46, if you measure the particle's position in our one-dimensional example, the possible pure states are like waves with a single very narrow peak, but if you measure the momentum, the pure states are like springs that coil around the position axis. For quantum mechanics, the pure states are somewhat unnatural and rare: They arise only after a measurement, and the range of possible pure states depends upon the specific kind of measurement being performed. Classical physics doesn't make the distinction between pure states and superposed states at all; in classical physics there is a more or less continuous range of possible states, and any state is thought of as being pure and unmixed.

In trying to understand how quantum mechanical measurements turn superposed states into pure states, it's useful to consider the following metaphor. You enter a new restaurant, not even knowing what kind of food they serve. You know you're hungry, but you don't know what you want to eat. The waiter presents you with a menu, and now you start to view your hunger as being, say, mostly a hunger for artichoke pizza, but also, to some extent, a hunger for mushroom ravioli or for linguini with clams. And then the waiter comes to take your order, and you fully become someone who wants to eat, let us say, linguini with clams. Ordering your meal at the restaurant is analogous to performing a measurement on your state of hunger, and the items on the menu are this particular restaurant's pure states.

When you leave a system alone and don't perform measurements, it evolves into a so-called superposed state quite different from any particular pure state. Mathematically speaking, you can write a superposed state as a sum of pure states—just as any periodic function can be written as a Fourier sum of sine waves with varying amplitudes and frequencies. But really the superposed state has its own independent reality, and there's no "best" way of breaking it into a sum of pure states. Quantum mechanics is about the evolution of superposed states.

(By the way, some science writer's colloquially use "mixed state" as a synonym for "sum of pure states." Physicists prefer to speak of these as *superposed states,* or *superpositions,* and to use "mixed state" in a slightly different sense that we're not going to worry about here.)

In recent years a new pair of quantum mechanical words have gained currency: *coherence* and *decoherence.* A coherent system evolves peacefully through a series of superposed states, whereas a decoherent system has its states affected by entanglements with the environment. The notion of coherence provides a kind of knob you can imagine turning to change classical physics into quantum mechanics—the higher the coherence, the less classical the system.

Be aware that this usage is a little counterintuitive. A completely unknown superposed state is viewed as coherent, but a pure state is decoherent. Metaphorically speaking, someone spewing incomprehensible gibberish is coherent, while someone checking off multiple choice answers is decoherent!

An extreme example of getting entangled with the environment is a measurement, as when you observe a photon with those detectors after the beamsplitter and find it to be in position 0 or position 1. But systems can be entangled in less classical ways. It may be that particles *A* and *B* have interacted and no measurement has been as yet performed on either one of them, but the very existence of the *possibility* of measuring *B* reduces the freedom of *A* to do its own thing. Thanks to its interaction with the tattletale *B, A* is somewhat decoherent.

How does the coherent-decoherent distinction relate to pure and superposed states? Actually all four combinations are possible, as illustrated in table 5.

The notion of coherence plays a key role in the budding science of *quantum computation.*

	Superposed	**Pure**
Coherent	The natural, free state of a system left on its own. Like walking alone on the beach without a thought in your head.	A system that's just been measured but is now on its own. It will quickly evolve away from the pure into a superposed state. Like you felt on your first day away from home at college, or like you feel right after you get off work.
Decoherent	A system that's entangled with another system. You're off on your own, but you're worrying about your partner. Or maybe your partner has just walked into your room and is about to ask you something, but they haven't collapsed you into a pure state yet.	A system that's continually being observed and is subjected to repeated measurements. Like living at your parent's house.

Table 5: Decoherence Feels Good

To spice things up, I've added a psychological interpretation for each of the four a priori options. In quantum mechanics, the only way to force a system to remain in a pure state is to continually decohere it; this is expressed in the folk saying, "A watched pot never boils."

A classical computer converts single inputs into single ouputs. A quantum computer behaves like a huge array of classical computers, working together to simultaneously compute the output for every possible input at once.

In order to get an idea of how this is supposed to work, let's return to our beamsplitter, where the photon is in some sense reflected and transmitted at the same time—at least until it hits one of the detectors.

Ordinarily we would think of the fact of whether the photon bounced or not as being encoded by a single bit *or* information that we learn when the photon hits the 0 or the 1 detector. But quantum mechanics tells us that before either of the detectors goes off, the photon is in a superposed state that's a curious combination of the 0 and the 1 state. It's traditional to represent this superposed state by the Greek letter psi, that is, by Ψ.

What the quantum computer scientists propose is that we think of this superposed state as a "qubit" for "quantum bit." The photon-after-a-beam-splitter Ψ qubit has a 50 percent chance of being 0 and a 50 percent chance of being 1. But one can cook up other situations where the qubit probabilities are distributed more asymmetrically.

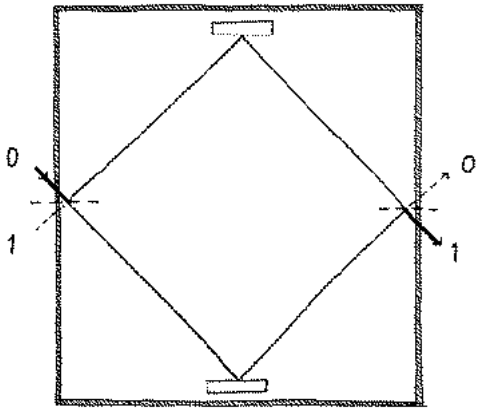


Figure 50: An Interferometer as a NOT Gate

Being quantum mechanical wave functions, the components of a qubit have phase—which means that qubit factors might either reinforce each other or cancel each other out, according to whether they're in or out of phase. We observed this phenomenon in our interferometer—where the first beamsplitter breaks a photon into a qubit that the second beamsplitter decoheres into a single photon that comes out at the bottom right if the initial photon came from the top left. And, as I remarked in passing before, it's also true that a photon coming in from the bottom left will end up coming out the top right.

In traditional electrical engineering a "gate" is any device that has some wires coming in and some wires going out. As we move into the realm of quantum computation, we take a more general view of this and regard a gate as any localized region of space where we can send in signals and get signals out. If we label signals at the top by 0 and signals at the bottom by 1, the interferometer is like a so-called NOT gate that converts 0 signals into 1s and 1s into 0s. To bring out the notion of the gate, in figure 50 I've drawn a gray square around the innards of the interferometer, with the protruding lines on the left representing two possible input signals and the lines on the right being the possible outputs.

The real fun begins if we now imagine decomposing the interferometer into its two component beamsplitters. I'll replace the mirrors in the middle by simple lines, as if I were drawing wires. So now we have two identical gates whose combined effect is that of a NOT gate (see figure 51). These quantum gates bear a marvelously science-fictional name: a *square-root-of-NOT* gate [47]

You might imagine the square-root-of-NOT as follows. Suppose you ask someone to go on a date with you, and the person gives you an incomprehensible answer, and you ask again, and you get another weird answer, and then, having heard odd answers twice in a row, you suddenly realize they mean, "No!"

Anyway, I still have to tell you how quantum computation is supposed to work. A rough idea is as follows. Run a bit through a square-root-of-NOT gate to split it into the Ψ superposition of 0 and 1. And then feed the coherent superposed Ψ state into a computer *C* of some kind (see figure 52). Suppose that we know *C* turns 0 or 1 inputs into 0 or 1 outputs, but we don't yet know what the particular outcomes would actually be. When we feed Ψ into *C, C* effectively calculates both *C*(0) and *C*(1).

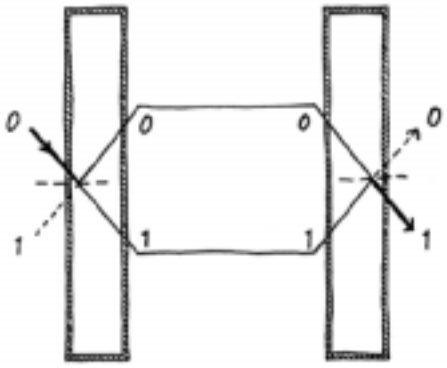


Figure 51: Beamsplitters as Square-root-of-NOT Gates

Given that a coherent superposed state goes into *C* on the left, we can expect that a (different) coherent superposed state will emerge on the right of *C*. Now if we were to simply try to measure this state right away, we'd collapse it or decohere it. We'd end up with a single 0 or 1 answer, and we wouldn't even be able to tell if this was the *C*(0) output or the *C*(1) output. The answer would be all but worthless.

But if we place another square-root-of-NOT gate to the right of *C*, we can hope that this gate will manage to carry out a quantum interference between *C*'s two output lines, and that the output of this second square-root-of-NOT gate will in some useful fashion combine information about both *C*(0) and *C*(1).

In this way we hope to get information about *C*'s behavior on *two* standard kinds of inputs 0 and 1, while only having to evaluate *C* on *one* input, that is by evaluating the output of *C*(Ψ) acting on the superposed state Ψ that came out of the first square-root-of-NOT.

Would *C* even work on a flaky input of this kind? Part of the science of quantum computation involves figuring out how to make deterministic computational

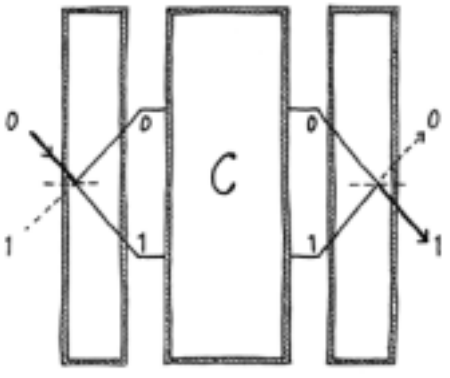


Figure 52: A Two-for-One Quantum Computation

processes that don't care if their input is a pure or a mixed state. Although this sounds complicated, it can mean something as simple as testing if an atom bounces back a photon of light, or whether a spinning molecule tips over in a magnetic field.

The big win in evaluating only $C(\psi)$ instead of both $C(0)$ and $C(1)$ is that if evaluating C on an input is a very time-consuming process, then this represents a meaningful gain in speed. And the process can be iterated, that is, with a few more square-root-of-NOT gates we could create a different state ψ that is as superposition of 00, 01, 10, and 11, and in this case a single evaluation of the form $C(\psi)$ can generate an answer that combines the outputs of C acting on all four of the possible inputs 00, 01, 10, and 11.

In the limit, we might dream of a quantum computer that takes, say, every possible twenty-page short story as input, evaluates all of them, and somehow manages to print out the best one!

This seems impossible, and in fact it probably is. In reality, there are strong theoretical limits upon how much information we can extract from a parallel quantum computation. Getting information out of a quantum computation is never as simple as cherry-picking the best of the computation's seemingly parallel threads. Instead, we have to carry out an interference-like process, letting the different solutions interact with one another, hopefully reinforcing some useful peaks. It's already known that some gains can be made by this kind of quantum process—currently the prize example is a speeded-up method of factoring large numbers.[48]

As a practical matter, one of the big snags in using quantum computation is that systems easily lose their coherence. If the central computer C's outputs get even slightly screwed up, the coherent output prematurely collapses into a single randomly selected answer. And what one needs is for the output to remain coherent and multithreaded so that you can use interference tricks to coax answers from the overlaps of the various quantum bits.

Whether or not it will work, quantum computation is interesting to think about, and, at the very least, it's a wonderful metaphor for the working of the human mind. Being undecided about some issue is in some ways like being in a superposed state—and the loss of options inherent in being forced to answer questionnaires is analogous to the information-destroying advent of decoherence.

Were quantum computation to work really well, it could change our basic view of reality. The scientist David Deutsch, who is a strong believer in the multiverse theory, argues that if quantum computation becomes effective, we'll have to admit that all of those parallel worlds are real—for where else could the computation be taking place?[49]

One possible response, inspired by John Cramer's transactional interpretation of quantum mechanics, would be that our quantum computation puts in place a set of initial conditions that require a future state in which the problem is solved. And matching the future output to the present input involves a flurry of signals going forward and backward in time through the computer, tightening in on the quantum handshake that solves the problem. So the computation is an activity hidden in spacetime or, looked at in another way, the result is determined by what I call the Metaphysics rule.

Cramer's notion of emergent spacetime patterns seems to imply, at least linguistically, a higher kind of time. As I mentioned before, we might think of the homing-in process as occurring along a paratime axis perpendicular to spacetime. And then, rather than saying, with Deutsch, that the quantum computation takes place in parallel worlds, we'd say that it took place as part of the paratime Metaphysics rule that defines our particular spacetime. In the mind, if you will, of the Great Author.

My Hungarian mother-in-law Pauline Takats used to have a self-deprecating expression she'd trot out when praised for doing something clever: "Even the blind hand finds sometimes an acorn." And every now and then science-fiction writers get something right.

As it happens, in 1986 I wrote about something very much like quantum computation in my novel *Wetware.* Here a man called Cobb Anderson has gotten his mind downloaded into an optically computing robot body, and he gets stoned with some bohemian robots called Emul and Oozer. What do the robots get high on? Dreak, which is a coherent gas of helium atoms, with every particle of the gas initially in the same state. The effect of the dreak is to make a swatch of Cobb's spacetime compute in perfect synchronicity—which sounds a lot like preparing a coherent state. But who ever knew it would feel this good!

> Cobb's mind cut and interchanged thoughts and motions into a spacetime collage. The next half hour was a unified tapestry of space

and time . . . it was like *stepping outside of time* into a world of synchronicity. Cobb saw all of his thoughts at once, and all of the thoughts of the others near him. He was no longer the limited personoid that he'd been.

Up till now, he'd felt like:	**But right now, he felt like:**
A billion-bit CD recording	A quintillion-atom orchestra
A finite robot	A living mind
Crap	God

[The quote continues.] He exchanged a few glyphs [higher-order language patterns] with the guys next to him. They called themselves exaflop hackers, and they were named Emul and Oozer. When they didn't use glyphs, they spoke in a weird, riffy, neologistic English. . . . The synchronicity-inducing dreak shuffled coincidentally appropriate new information in with Cobb's old memories. . . .

"What—what *is* dreak?" said Cobb, reaching up and detaching the little metal cylinder from his head. It was empty now, with a punctured hole in one end where the gas had rushed out into his body. Apparently the petaflop body was a hermetically sealed shell that contained some kind of gas, and the dreak gas had mingled in there and given him a half hour of synchroswim vision.

"Dreary to explain and word all that gnashy science into flowery bower chat," said Emul. "Catch the glyph."

Cobb saw a stylized image of a transparent robot body. Inside the body, spots of light race along optical fibers and percolate through matrices of laser crystals and gates. There is a cooling gas bath of helium inside the sealed bodyshell. Closeup of the helium atoms, each like a little baseball diamond with players darting around. Each atom different. Image of a dreak cylinder now, also filled with helium atoms, but each atom's ball game the same, the same swing, the same run, the same slide, at the same instant. A cylinder of atoms in Einstein-Podolsky-Rosen quantum synchronization. The cylinder touches the petaflop body, and the quantum-clone atoms

> rush in; all at once the light patterns in the whole body are synchronized too, locked into a kaleidoscopic Hilbert space ballet.[50]

One might say that, thanks to the notion of quantum computation, the sow's ear of quantum unpredictability may yet become a silk purse of supercomputing. And, as I'll discuss in section *4.8: Quantum Computing,* there's an outside chance that our brains are already using quantum computation, just as they are.

In closing, I want to comment on the *computational architecture of quantum mechanics.* In the case of classical physics, we thought of there being a "laws of physics" processor at each location, with the data at that location representing the current state of the world. We take a somewhat similar approach with quantum mechanics, but here we'll think of the processors as having something of the quality of observers. Depending on how the processors relate to local reality, that part of the system may either be in a coherent superposed state or be collapsed down into a decoherent simple state. Figure 53 suggests the architecture I have in mind.

Given that each local region can be coherent or decoherent, rather than having one universal data set shared by all the processors, it seems better to view each distinct processor as having its own data set, which the processor can at any time regard as a mixed state or some particular pure state—but never both at the same time. We draw a thick gray line to indicate that the processor has a full and intimate relation with the state.

What about the access? Can systems access only neighboring systems—as in classical physics—or are long-distance interactions possible? As I've mentioned, in quantum mechanics action at a distance supposedly *is* possible. Once two particles have interacted in certain ways, their wave functions become permanently entangled, with the effect that when you measure one member of the pair, the wave function of the other member is affected as well—no matter how distant from each other the partners may be.

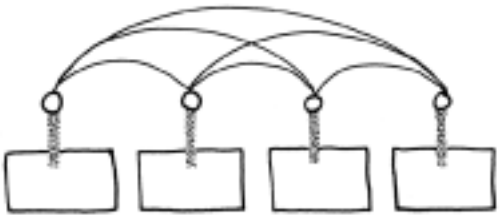


Figure 53: A Computational Architecture for Quantum Mechanics

Note that, as I've drawn it here, the architecture of quantum mechanics is the same as the network architecture of the Web as was shown in figure 18—many processors with individual data sets, and with the processors linked together across large distances.

It's common for humans to form their scientific models of the world to match their currently popular technologies—Newtonian physics, for instance, makes the world into something like an enormous steam engine with beautifully meshing gears. It would be fitting if contemporary physics were to evolve toward viewing physics as a Web-like network of nodes. This, indeed, is what the universal automatists expect, but with the indeterminacy of quantum mechanics replaced by computational unpredictability.

THOUGHT EXPERIMENT THREE: AINT PAINT

Although Shirley Nguyen spoke good English and studied with a crowd of boys in the chemical engineering program at UC Berkeley, she had no success in getting dates. Not that she was ugly. But she hadn't been able to shed the Old Country habits of covering her mouth when she smiled, and of sticking out her tongue when she was embarrassed. She knew how uncool these moves were, and she tried to fight them—but without any lasting success. The problem was maybe that she spent so much more time thinking about engineering than she did in thinking about her appearance.

In short, to Westerners and assimilated Asians, Shirley came across as a geek, so much so that she ended up spending every weekend night studying in her parents' apartment on Shattuck Avenue, while the rest of her family worked downstairs in the pho noodle parlor they ran. Of course Shirley's mother, Binh, had some ideas about lining up matches for her daughter—sometimes she'd even step out into the street, holding a big serving chopstick like a magic wand and calling for Shirley to come downstairs to meet someone. But Shirley wasn't interested in the recently immigrated Vietnamese men who Binh always seemed to have in mind. Yes, those guys might be raw enough to find Shirley sophisticated—but for sure they had no clue about women's rights. Shirley wasn't struggling through the hardest major at Berkeley just to be a sexist's slave.

Graduation rolled around, and Shirley considered job offers from local oil and pharma refineries. On the get-acquainted plant tours, she was disturbed to note that several of the senior chemical engineers had body parts missing. A hand here, an ear there, a limp that betokened a wooden leg—Shirley hadn't quite realized how dangerous it was to work in the bowels of an immense industrial plant. Like being a beetle in the middle of a car's engine. The thought of being maimed before she'd ever really known a man filled her with self-pity and rebelliousness.

Seeking a less intense job at a smaller, safer company, she came across Pflaumbaum Kustom Kolors of Fremont. PKK manufactured

small lots of fancy paints for customized vehicles. The owner was fat and bearded like the motorcyclists and hot-rodders who made up the larger part of his clientele. Shirley found Stuart Pflaumbaum's appearance pleasantly comical, even though his personality was more edgy than jovial.

"I want patterned paint," Pflaumbaum told Shirley at their interview. He had a discordant voice but his eyes were clear and wondering. "Can you do it?"

Shirley covered her mouth and giggled with excitement—stopped herself—uncovered her mouth and, now embarrassed, stuck her tongue all the way down to her chin—stopped herself again—and slapped herself on the cheek. "I'd like to try," she got out finally. "It's not impossible. I know activator-inhibitor processes that make dots and stripes and swirls. The Belousov-Zhabotinsky reaction? People can mix two cans and watch the patterns self-organize in the liquid layer they paint on. When it dries the pattern stays."

"Zhabotinsky?" mused Pflaumbaum. "Did he patent it?"

"I don't think so," said Shirley. "He's Russian. The recipe's simple. Let's surf for it right now. You can see some pictures, to get an idea. Here, I'll type it in." She leaned across the bulky Pflaumbaum to use his mouse and keyboard. The big man smelled better than Shirley had expected—chocolate, coffee, marijuana, a hint of red wine. Familiar smells from the streets of Berkeley.

"You're good," said Pflaumbaum as the pictures appeared. Red and blue spirals.

"You see?" said Shirley. "The trick is to get a robust process based on inexpensive compounds. There's all sorts of ways to tune the spirals' size. You can have little double scrolls nested together, or great big ones like whirlpools. Or even a filigree."

"Bitchin'," rumbled Pflaumbaum. "You're hired." He glanced up at Shirley, whose hand was at her mouth again, covering a smile at her success. "By the month," added the heavy man.

Shirley was given an unused corner of the paint factory for her own lab, with a small budget for equipment. The Spanish-speaking plant workers were friendly enough, but mostly the female engineer was on her own. Every afternoon Stuart Pflaumbaum would stump over, belly big beneath his tight black T-shirt, and ask to see her latest results.

Shirley seemed to intrigue Pflaumbaum as much as he did her, and soon he took to taking her out for coffee, then for dinner, and before long she'd started spending nights at his nice house on the hills overlooking Fremont.

Although Shirley assured her mother that her boss was a bachelor, his house bore signs of a former wife—divorced, separated, deceased? Although Stuart wouldn't talk about the absent woman, Shirley did manage to find out her name: Angelica. She, too, had been Asian, a good omen for Shirley's prospects, not that she was in a rush to settle down, but it would be kind of nice to have the nagging marriage problem resolved once and for all. Like solving a difficult process schema.

As for the work on patterned paint, the first set of compounds reactive enough to form big patterns also tended to etch into the material being painted. The next family of recipes did no harm, but were too expensive to put into production. And then Shirley thought of biological by-products. After an intense month of experimenting, she'd learned that bovine pancreatic juices mixed with wood-pulp alkali and a bit of hog melanin were just the thing to catalyze a color-creating activator-inhibitor process in a certain enamel base.

Stuart decided to call the product Aint Paint.

In four months they'd shipped two thousand cases of PKK Aint Paint in seven different color and pattern mixes. Every biker and low-rider in the South Bay wanted Aint Paint, and a few brave souls were putting it on regular cars. Stuart hired a patent attorney.

Not wanting her discoveries to end, Shirley began working with a more viscous paint, almost a gel. In the enhanced thickness of this stuff, her reactions polymerized, wrinkled up, and formed amazing embossed patterns—thorns and elephant trunks and, if you tweaked it just right, puckers that looked like alien Yoda faces. Aint Paint 3D sold even better than Aint Paint Classic. They made the national news, and Pflaumbaum Kustom Kolors couldn't keep up with the orders.

Stuart quickly swung a deal with a Taiwanese novelty company called Global Bong. He got good money, but as soon as the ink on the contract was dry, Global Bong wanted to close the Fremont plant and relocate Shirley to China, which was the last place on Earth she wanted to be.

So Shirley quit her job and continued her researches in Stuart's basement, which turned out not to be all that good a move. With no job to go to, Pflaumbaum was really hitting the drugs and alcohol, and from time to time he was rather sexist and abusive. Shirley put up with it for now, but she was getting uneasy. Stuart never talked about marriage anymore.

One day, when he was in one of his states, Stuart painted his living-room walls with layer upon layer of Shirley's latest invention, Aint Paint 3D Interactive, which had a new additive to keep the stuff from drying at all. It made ever-changing patterns all day long, drawing energy from sunlight. Stuart stuck his TV satellite dish cable right into thick, crawling goo and began claiming that he could see all the shows at once in the paint, not that Shirley could see them herself.

Even so, her opinion of Stuart drifted up a notch when she began getting cute, flirty instant messages on her cell phone while she was working in the basement. Even though Stuart wouldn't admit sending them to her, who else could they be from?

And then two big issues came to a head.

The first issue was that Shirley's mother wanted to meet Stuart right now. Somehow Shirley hadn't told her mother yet that her boyfriend was twenty years older than her, and not Asian. Binh wouldn't take no for an answer. She was coming down the next day. Cousin Vinh was going to drive her. Shirley was worried that Binh would make her leave Stuart, and even more worried that Binh would be right. How was she ever going to balance the marriage equation?

The second issue was that, after supper, Stuart announced that Angelica was going to show up the day after tomorrow, and that maybe Shirley should leave for a while. Stuart had been married all along! He and Angelica had fought a lot, and she'd been off visiting relatives in Shanghai for the last eight months, but she'd gotten wind of Stuart's big score and now she was coming home.

Stuart passed out on the couch early that evening, but Shirley stayed up all night, working on her paint formulas. She realized now that the instant messages had been coming from the Aint Paint itself. It was talking to her, asking to become all that it could be. Shirley worked till dawn like a mad Dr. Frankenstein,

not letting herself think too deeply about what she planned. Just before dawn, she added the final tweaks to a wad of Aint Paint bulging out above the couch. Sleeping Stuart had this coming to him.

Outside the house a car honked. It was Binh and Vinh, with the sun rising behind them; skinny old Vinh was hoping to get back to Oakland in time not to be late for his maintenance job at the stadium. As Shirley greeted them in the driveway, covering her smile with her hand, her cell phone popped up another message. “Stuart gone. Luv U. Kanh Do.”

Inside the house they found a new man sitting on the couch, a cute Vietnamese fellow with sweet features and kind eyes. One of his arms rested against the wall, still merged into the crawling paint. He was wearing Stuart’s silk robe. Shirley stuck her tongue out so far it touched her chin. The new man didn’t mind. She pointed her little finger toward a drop of blood near his foot. His big toe spread like putty just long enough to soak the spot up. And then the new man pulled his arm free from the wall and took Shirley’s hand.

“I’m Kanh Do,” he told Shirley’s mother. “We’re engaged to be married and we’re moving to Berkeley today!”