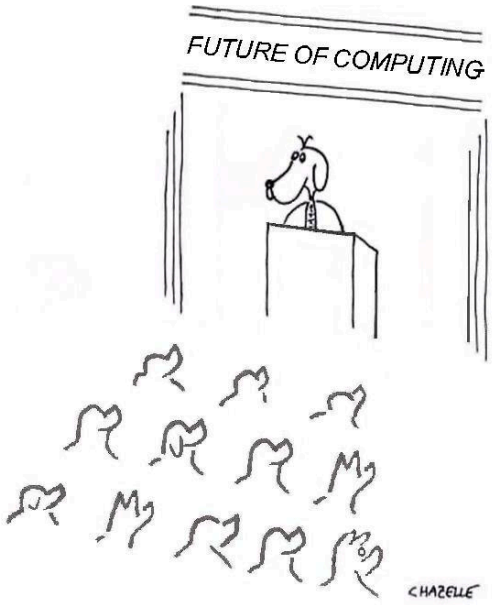
The Algorithm: Idiom of Modern Science

by Bernard Chazelle

When the great Dane of 20th century physics, Niels Bohr, was not busy chewing on a juicy morsel of quantum mechanics, he was known to yap away witticisms worthy of Yogi Berra. The classic Bohrism "*Prediction is difficult, especially about the future*" alas came too late to save Lord Kelvin. Just as physics was set to debut in Einstein's own production of *Extreme Makeover*, Kelvin judged the time ripe to pen the field's obituary: "*There is nothing new to be discovered in physics now.*" Not his lordship's finest hour.

Nor his worst. Aware that fallibility is the concession the genius makes to common mortals to keep them from despairing, Kelvin set early on to give the mortals much to be hopeful about. To wit, the thermodynamics pioneer devoted the first half of his life to studying hot air and the latter half to blowing it. Ever the perfectionist, he elevated to an art form the production of pure, unadulterated bunk: "*X-rays will prove to be a hoax*"; "*Radio has no future*"; "*Heavier-than-air flying machines are impossible*"; and my personal favorite, "*In science there is only physics; all the rest is stamp collecting.*" Kelvin's crystal ball was the gift that kept on giving.

Gloat not at a genius' misfortunes. Futurologitis is an equal-opportunity affliction, one hardly confined to the physicist's ward. "*I think there is a*



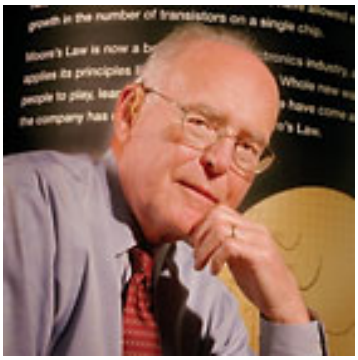
" Soon, my friends, you will look at a child's homework — and see nothing to eat. "

world market for maybe five computers," averred IBM Chairman, Thomas Watson, a gem of prescience matched only by a 1939 New York Times editorial: "*The problem with television is that people must sit and keep their eyes glued to the screen; the average American family hasn't time for it.*" The great demographer Thomas Malthus owes much of his fame to his loopy prediction that exponentially increasing populations would soon outrun the food supply. As the apprentice soothsayer learns in "Crystal Gazing 101," never predict a geometric growth!

Apparently, Gordon Moore skipped that class. In 1965, the co-founder of semiconductor giant Intel announced his celebrated law: *Computing power doubles every two years.* Moore's Law has, if anything, erred on the conservative side. Every eighteen months, an enigmatic pagan ritual will see white-robed sorcerers silently shuffle into a temple dedicated to the god of cleanliness, and soon reemerge with, on their faces, a triumphant smile and, in their hands, a silicon wafer twice as densely packed as the day before. No commensurate growth in human mental powers has been observed: this has left us scratching our nonexpanding heads, wondering what it is we've done to deserve such luck.

To get a feel for the magic, consider that the latest Sony PlayStation would easily outpace the fastest supercomputer from the early nineties. If not for Moore's Law, the Information Superhighway would be a back alley to Snoozeville; the coolest thing about the computer would still be the blinking lights. And so, next time you ask who engineered the digital revolution, expect many hands to rise. But watch the long arm of Moore's Law tower above all others. Whatever your brand of high-tech addiction, be it IM, iPod, YouTube, or Xbox, be aware that you owe it first and foremost to the engineering wizardry that has sustained Moore's predictive prowess over the past forty years.

Enjoy it while it lasts, because it won't. Within a few decades, say the optimists, a repeal is all but certain. Taking their cue from Bill Gates, the naysayers conjure up the curse of power dissipation, among other woes, to declare Moore's Law in the early stage of rigor mortis. Facing the bleak, sorrowful tomorrows of ♫*The Incredible Shrinking Chip That Won't Shrink No More,*♫ what's a computer scientist to do?



The rule of law

Break out the Dom and pop the corks, of course! Moore's Law has added fizz and sparkle to the computing cocktail, but for too long its exhilarating potency has distracted the party-goers from their Holy Grail quest: *How to unleash the full computing and modeling power of the Algorithm*. Not to stretch the metaphor past its snapping point, the temptation is there for the *Algorithmistas* (my tribe) to fancy themselves as the Knights of the Round Table and look down on Moore's Law as the Killer Rabbit, viciously elbowing King Arthur's intrepid algorithmic warriors. Just as an abundance of cheap oil has delayed the emergence of smart energy alternatives, Moore's Law has kept algorithms off center stage. Paradoxically, it has also been their enabler: the killer bunny turned sacrificial rabbit who sets the track champion on a world record pace, only to fade into oblivion once the trophy has been handed out. With the fading imminent, it is not too soon to ask why the Algorithm is destined to achieve celebrity status within the larger world of science. While you ask, let me boldly plant the flag and bellow the battle cry:

> "*The Algorithm's coming-of-age as the new language of science promises to be the most disruptive scientific development since quantum mechanics.*"

If you think such a blinding flare of hyperbole surely blazed right out of Lord Kelvin's crystal ball, read on and think again. A computer is a storyteller and algorithms are its tales. We'll get to the tales in a minute but, first, a few words about the storytelling.

Computing is the meeting point of three powerful concepts: *universality*, *duality*, and *self-reference*. In the modern era, this triumvirate has bowed to the class-conscious influence of the *tractability* creed. The creed's incessant call to complexity class warfare has, in turn, led to the emergence of that ultimate class leveler: *the Algorithm*. Today, not only is this new "order" empowering the e-technology that stealthily rules our lives; it is also challenging what we mean by knowing, believing, trusting, persuading, and learning. No less. Some say the Algorithm is poised to

become the new *New Math*, the idiom of modern science. I say The Sciences They Are A-Changin' and the Algorithm is Here to Stay.

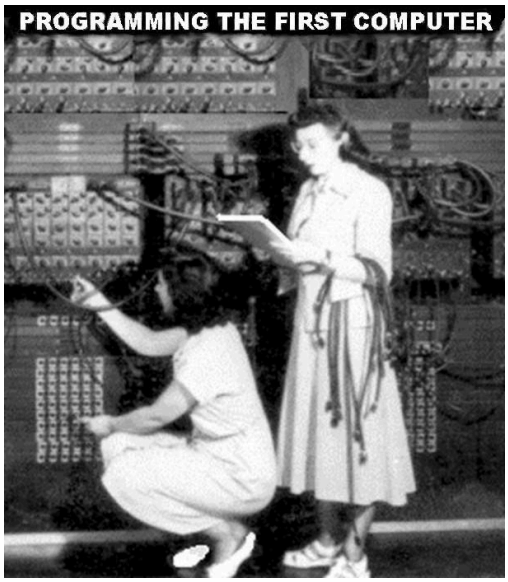
Reread the previous paragraph. If it still looks like a glorious goulash of blathering nonsense, good! I shall now explain, so buckle up!

The universal computer

Had Thomas Jefferson been a computer scientist, school children across the land would rise in the morning and chant these hallowed words:

> "*We hold these truths to be self-evident, that all computers are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Universality and the separation of Data, Control, and Command.*"

Computers come in different shapes, sizes, and colors, but all are created equal—indeed, much like 18th century white male American colonists. Whatever the world's fastest supercomputer can do (in 2006, that would be the IBM Blue Gene/L), your lowly iPod can do it, too, albeit a little more slowly. Where it counts, size doesn't matter: all computers are qualitatively the same. Even exotic beasts such as quantum computers, vector machines, DNA computers, and cellular automata can all be viewed as fancy iPods. That's *universality!*



The field of computing later opened up to men

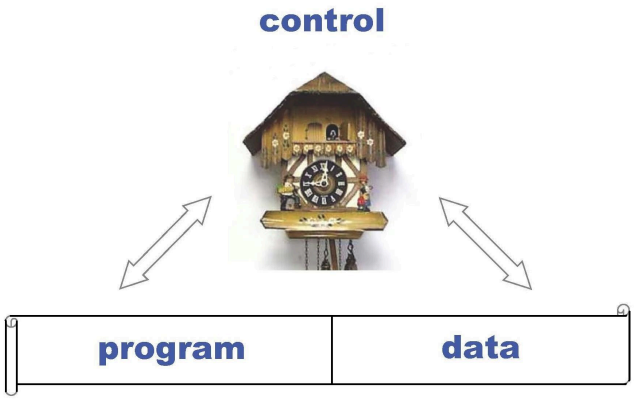
Here's how it works. Your iPod is a tripod (where did you think they got that name?), with three legs called *control*, *program*, and *data*. Together, the program and the data form the two sections of a document [*program* | *data*] that, to the untrained eye, resembles a giant, amorphous string of 0s and 1s. Something like this:

[*11100010101100100 10* | *10101110101010011 10*]

Each section has its own, distinct purpose: the program specifies instructions for the control to follow (eg, how to convert text into pdf); the

data encodes plain information, like this essay (no, not plain in *that* sense). The data string is to be read, not to be read *into*. About it, Freud would have quipped: "*Sometimes a string is just a string*." But he would have heard, seeping from the chambers of a program, the distant echoes of a dream: jumbled signs crying out for interpretation. To paraphrase the Talmudic saying, an uninterpreted program is like an unread letter. The beauty of the scheme is that the control need not know a thing about music. In fact, simply by downloading the appropriate program-data document, you can turn your iPod into: an earthquake simulator; a word processor; a web browser; or, if downloading is too much, a paperweight. Your dainty little MP3 player is a *universal* computer.

The control is the computer's brain and the sole link between program and data. Its only function in life is to read the data, interpret the program's orders, and act on them—a task so pedestrian that modern theory of reincarnation ranks the control as the lowest life form on the planet, right behind the inventor of the CD plastic wrap. If you smash your iPod open with a mallet and peek into its control, you'll discover what a marvel of electronics it is—okay, was. Even more marvelous is the fact that it need not be so. It takes little brainpower to follow orders blindly (in fact, too much tends to get in the way). Stretching this principle to the limit, one can design a universal computer with a control mechanism so simple that any old cuckoo clock will outsmart it. This begs the obvious question: did Orson Welles know that when he dissed the Swiss and their cuckoo clocks in "The Third Man"? It also raises a suspicion: doesn't the control need to add, multiply, divide, and do the sort of fancy footwork that would sorely test the nimblest of cuckoo clocks?



Alas, it still strikes the hours

No. The control on *your* laptop might indeed do all of those things, but the point is that it need *not* do so. (Just as a bank might give you a toaster when you open a new account, but it need *not* be a toaster; it could be a pet hamster.) Want to add? Write a program to add. Want to divide? Write a program to divide. Want to print? Write a program to print. A control that delegates all it can to the program's authority will get away with a mere two dozen different "states"—simplicity a cuckoo clock could only envy. If you want your computer to do something for you, don't just scream at the control: write down instructions in the program section. Want to catch trouts? Fine, append a fishing manual to the program

string. The great nutritionist Confucius said it better: "*Give a man a fish and you feed him for a day. Teach a man to fish and you feed him for a lifetime.*" The binary view of fishing = river + fisherman makes way for a universal one: fishing = river + fishing manual + you. Similarly,

computing = data + program + control.

This tripodal equation launched a scientific revolution, and it is to British mathematician Alan Turing that fell the honor of designing the launching pad. His genius was to let robots break out of the traditional binary brain-brawn mold, which conflates control and program, and embrace the liberating "tripod-iPod" view of computing. Adding a third leg to the robotic biped ushered in the era of universality: any computer could now simulate any other one.

Underpinning all of that, of course, was the *digital* representation of information: DVD vs VCR tape; piano vs violin; Anna Karenina vs Mona Lisa. The analog world of celluloid film and vinyl music is unfit for reproduction: doesn't die; just fades away. Quite the opposite, encoding information over an alphabet opens the door to unlimited, decay-free replication. In a universe of 0s and 1s, we catch a glimpse of immortality; we behold the gilded gates of eternity flung wide open by the bewitching magic of a lonely pair of incandescent symbols. In short, analog sucks, digital rocks.

Two sides of the same coin

Load your iPod with the program-data document [*Print this* | *Print this*]. Ready? Press the start button and watch the words "*Print this*" flash across the screen. Exciting, no? While you compose yourself with bated breath amid the gasps and the shrieks, take stock of what happened. To the unschooled novice, data and program may be identical strings, but to the cuckoo-like control they couldn't be more different: the data is no more than what it *is*; the program is no less than what it *means*. The control may choose to look at the string "*Print this*" either as a meaningless sequence of letters or as an order to commit ink to paper. To scan symbols mulishly or to deforest the land: that is the option at hand here—we call it *duality*.

So 1907, I almost hear you sigh. In that fateful year, Ferdinand de Saussure, the father of linguistics, announced to a throng of admirers that there are two sides to a linguistic sign: its *signifier* (representation) and its *signified* (interpretation). A string is a sign that, under the watchful

eye of the control, acts as signifier when data and as signified when a program.

Saussure's intellectual progeny is a breed of scholars known as semioticians. Funny that linguists, of all people, would choose for themselves a name that rhymes with mortician. Funny or not, semiotics mavens will point out the imperfect symmetry between program and data. The latter is inviolate. Signifiers must be treated with the utmost reverence: they could be passwords, hip-hop rhymes, or newfound biblical commandments. Mess with them at your own peril.

Programs are different. The encoding of the signified is wholly conventional. Take the program "*Print this*", for example. A francophonic control would have no problem with "*Imprimer ceci* " or, for that matter, with the obsequious "*O, control highly esteemed, may you, noblest of cuckoos, indulge my impudent wish to see this humble string printed out, before my cup runneth over and your battery runneth out*." The plethora of programming languages reveals how so many ways there are of signifying the same thing. (Just as the plethora of political speeches reveals how so many ways there are of signifying nothing.)

Sensing the comic, artistic, and scholarly potential of the duality between program and data, great minds went to work. Abbott and Costello's "Who's on First?" routine is built around the confusion between a baseball player's nickname (the signifier) and the pronoun "who" (the signified). Magritte's celebrated painting "*Ceci n'est pas une pipe*" (this is not a pipe) plays on the distinction between the picture of a pipe (the signifier) and a pipe one smokes (the signified). The great painter might as well have scribbled on a blank canvas: "*Le signifiant n'est pas le signifié* " (the signifier is not the signified). But he didn't, and for that we're all grateful.

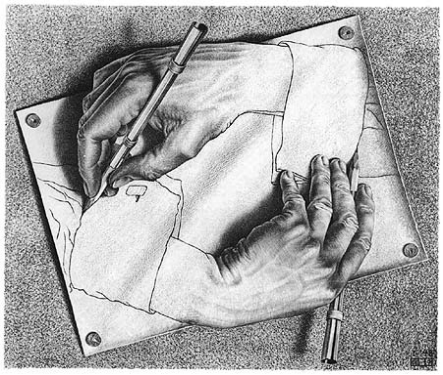
English scholars are not spared the slings and arrows of duality either. How more dual can it get than the question that keeps Elizabethan lit gurus awake at night: "Did Shakespeare write Shakespeare?" And pity the dually tormented soul that would dream up such wacky folderol: "*Twas brillig, and the slithy toves Did gyre and gimble in the wabe; All mimsy were the borogoves, And the mome raths outgrabe*."

Say it ain't true

I am lying. Really? Then I am lying when I say I am lying; therefore, I am not lying. Yikes. But if I am not lying then I am not lying when I say I am lying; therefore, I am lying. Double yikes. Not enough yet? Okay, consider the immortal quip of the great American philosopher Homer Simpson: "*Oh Marge, cartoons don't have any deep meaning; they're just stupid drawings that give you a cheap laugh.*" If cartoons don't have meaning, then Homer's statement is meaningless (not merely a philosopher, the man *is* a cartoon character); therefore, for all we know, cartoons *have* meaning. But then Homer's point is... Doh! Just say it ain't true. Ain't true? No, please, don't say it ain't true! Because if it ain't true then ain't true ain't true, and so...

AAARRRGGGHHH !!!!!!

Beware of *self-referencing*, that is to say, of sentences that make statements about themselves. Two of the finest mathematical minds in history, Cantor and Gödel, failed to heed that advice and both went stark raving bonkers. As the Viennese gentleman with the shawl-draped couch already knew, self-reference is the quickest route to irreversible dementia.



Escher's reproductive parts

It is also the salt of the computing earth. Load up your iPod, this time with the program-data document [*Print this twice | Print this twice*]. Push the start button and see the screen light up with the words: "*Print this twice Print this twice*". Lo and behold, the thing prints itself! Well, not quite: the vertical bar is missing. To get everything right and put on fast track your career as a budding computer virus artist, try this instead: [*Print this twice, starting with a vertical bar the second time | Print this twice, starting with a vertical bar the second time*]. See how much better it works now! The key word in the self-printing business is "twice": "never" would never work; "once" would be once too few; "thrice"?? Please watch your language.

Self-reproduction requires a tightly choreographed dance between: (i) a program explaining how to copy the data; (ii) a data string describing that very same program. By duality, the same sequence of words (or bits) is interpreted in two different ways; by self-reference, the duality coin looks the same on both sides. Self-reference—called *recursion* in computer parlance—requires duality; not the other way around. Which is why the universal computer owes its *existence* to duality and its *power* to recursion. If Moore's Law is the fuel of Google, recursion is its engine.

The tripodal view of computing was the major insight of Alan Turing—well, besides this little codebreaking thing he did in Bletchley Park that helped win World War II. Not to discount the lush choral voices of Princeton virtuosos Alonzo Church, Kurt Gödel, and John von Neumann, it is Maestro Turing who turned into a perfect opus the hitherto disjointed scores of the computing genre.

Mother Nature, of course, scooped them all by a few billion years. Your genome consists of two parallel strands of DNA that encode all of your genetic inheritance. Your morning addiction to Cocoa Puffs, your night cravings for Twinkies? Yep, it's all in there. Now if you take the two strands apart and line them up, you'll get two strings about three billion letters long. Check it out:

[*ACGGTATCCGAATGC...* | *TGCCATAGGCTTACG...*]

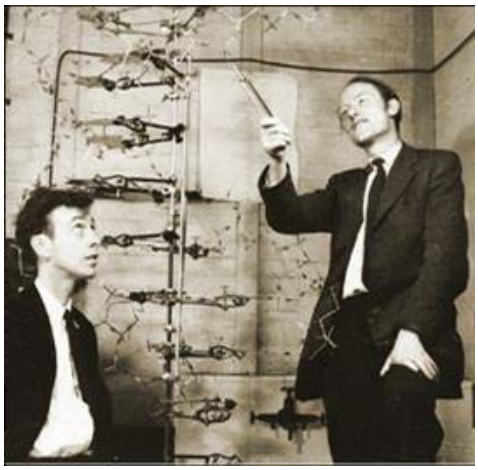
There they are: two twin siblings locking horns in a futile attempt to look different. Futile because if you flip the As into Ts and the Cs into Gs (and vice versa) you'll see each strand morph into the other one. The two strings are the same in disguise. So flip one of them to get a more symmetric layout. Like this:

[*ACGGTATCCGAATGC...* | *ACGGTATCCGAATGC...*]

Was I the only one to spot a suspicious similarity with [*Print this twice* | *Print this twice*] or did you, too? Both are program-data documents that provide perfectly yummy recipes for self-reproduction. Life's but a walking shadow, said Macbeth. Wrong. Life's but a self-printing iPod! Ministry-of-Virtue officials will bang on preachily about there being more to human life than the blind pursuit of self-replication, a silly notion that Hollywood's typical fare swats away daily at a theater near you. Existential angst aside, the string "*ACGGTATCCGAATGC...*" is either plain data (the genes constituting your DNA) or a program whose execution produces, among other things, all the proteins needed for DNA replication, plus all of the others needed for the far more demanding task of sustaining your Cocoa Puffs addiction. Duality is the choice you have to think of your genome either as a long polymer of nucleotides (the data to be read) or as the sequence of amino acids forming its associated proteins (the "programs of life"). Hence the fundamental equation of biology:

Life = Duality + Self-reference

On April 25, 1953, the British journal *Nature* published a short article whose understated punchline was the shot heard 'round the world: "*It has not escaped our notice*



" Elementary, my dear Watson! "

that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material." In unveiling to the world the molecular structure of DNA, James Watson and Francis Crick broke the Code of Life. In so doing, they laid bare the primordial link between life and computing. One can easily imagine the reaction of that other codebreaker from Bletchley Park: "*Duality and self-reference embedded in molecules? Jolly good to know God thinks like me.*"

Turing's swagger would have been forgivable. After all, here was the man who had invented the computer. Here was the man who had put the mind-matter question on a scientific footing. Here was the man who had saved Britain from defeat in 1941 by breaking the Nazi code. Alas, good deeds rarely go unpunished. In lieu of a knighthood, a grateful nation awarded Alan Turing a one-way ticket to Palookaville, England: a court conviction for homosexuality with a sentence of forced estrogen injections. On June 7, 1954, barely one year to the day of Watson and Crick's triumph, Alan Turing went home, ate an apple laced with cyanide, and died. His mother believed, as a mother would, that it was an accident.

The modern era

The post-Turing years saw the emergence of a new computing paradigm: *tractability*. Its origin lay in the intuitive notion that checking a proof of Archimedes's theorem can't be nearly as hard as finding it in the first place; enjoying a coke must be simpler than discovering its secret recipe (or so the Coca Cola Company hopes), falling under the spell of *'Round Midnight* ought to be easier than matching Monk's composing prowess. But is it really? Amazingly, no one knows. Welcome to the foremost open question in all of computer science!

Ever wondered whether the 1,000-song library stored in your iPod could be reordered and split up to form two equal-time playlists? Probably not. But suppose you wanted to transfer your songs to the two sides of an extra-length cassette while indulging your lifelong passion for saving money on magnetic tape. Which songs would you put on which side so as

to use as little tape as possible? Now you'd be wondering, wouldn't you? (Humor me: say yes.)

"First you prove it, then you let it sink in."

You wouldn't wonder long, anyway. After a minute's reflection, you'd realize you didn't have the faintest idea how to do that. (Warning: splitting a tune in the middle is a no-no.) Of course, you could try all possibilities but that's a big number—roughly 1 followed by 300 zeroes. Ah, but your amazing friend Alice, she knows! Or so she says. Then why not just get the two playlists from her? By adding up a few numbers, you'll easily verify that she's not lying and that, indeed, both lists have the same playing time. What Alice will hand you over is, in essence, a *proof* that your song library can be split evenly. Your job will be reduced to that of *proof-checking*, a task at which a compulsive tape-saving Scrooge might even shine. Heads-up: did you notice my nonchalant use of the word "lying"? When a movie's opening scene casually trains the camera on a gun, no one might get hurt for a while, but you know that won't last.

Alas, wondrous Alice fell down the rabbit hole eons ago and, these days, a good library splitting friend is hard to find. And so, sadly, you'll have little choice but to compile the two lists yourself and engage in that dreaded thing called *proof-finding*. That's a tougher nut to crack. So much so that even if you were to harness the full power of an IBM Blue Gene/L running the best software available anywhere on earth and beyond, the entire lifetime of the universe wouldn't be enough! You might get lucky with the parameters and get it done sooner, but getting lucky? Yeah, right...

To add insult to injury, computer scientists have catalogued thousands of such *Jurassic* problems—so named for the dinosaur-like quality of their solutions: hard to discover but impossible to miss when they pop up in front of you; in other words, proofs hopelessly difficult to find but a breeze to verify. Courtesy of Murphy's Law, of course, the great Jurassics of the world include all of the hydra-headed monsters we're so desperate to slay: drug design; protein folding; resource allocation; portfolio optimization; suitcase packing; etc. Furthermore, even shooting for good approximate solutions—when the notion makes sense—can sometimes be just as daunting.

Now a funny thing happened on the way back from the word factory. Despite its dazzling lyricism, metaphorical felicity, hip-hoppish élan, not to mention a Niagara of adulatory gushing I'll kindly spare you, my staggeringly brilliant coinage “Jurassic” hasn't caught on. Yet. Skittish computer scientists tend to favor the achingly dull “NP-complete.” Worse, their idea of bustin' a dope, def funky rhyme is to—get this—write down the thing in full, as in “complete for nondeterministic polynomial time.” To each their own.

Back to the Jurassics. Always basking in the spotlight, they are famously difficult, impossibly hard to satisfy, and—if their resilience is any guide—quite pleased with the attention. These traits often run in the family; sure enough, the Jurassics are blood kin. The first to put them on the analyst's couch and pin their intractable behavior on consanguinity were Stephen Cook, Jack Edmonds, Richard Karp, and Leonid Levin. In the process they redefined computing around the notion of *tractability* and produced the most influential milestone in post-Turing computer science.

But what is a tractable problem, you ask? Answer: one that can be solved in *polynomial time*. Oh, swell; nothing like calling upon the opaque to come to the rescue of the obscure! Relax: it's quite simple, really. If you double the size of the problem—say, your iPod library will now hold 2,000 tunes instead of a mere 1,000—then the time to find an even split should at most double, or quadruple, or increase by some *fixed* rate (ie, independent of the problem size). That's what it means to be *tractable*. Convoluted as this definition may seem, it has two things going for it: one is to match our intuition of what can be solved in practice (assuming the fixed rate isn't “fixed” too high); the other is to leave the particular computer we're working on out of the picture. See how there is no mention of computing speeds; only of growth rates. It is a statement about software, not hardware. Tractability is a *universal* attribute of a problem—or lack thereof. Note: some scholars prefer the word *feasibility*. Obviously, to resist the lure of the opening riff of Wittgenstein's “Tractatus Logico-Philosophicus” takes willpower; predictably, the feasibility crowd is thin.

“ What do you mean, ‘intractable’ ? ”

Library splitting does not appear to be tractable. (Hold the tears: you'll need them in a minute.) Any algorithm humans have ever tried—and many have—requires *exponential* time. Read: all of them share the dubious distinction that their running times get *squared*

(not merely scaled up by a constant factor) whenever one doubles the size of the problem. If you do the math, you'll see it's the sort of growth that quickly gets out of hand.

Well, do the math. Say you want to solve a problem that involves 100 numbers and the best method in existence takes one second on your laptop. How long would it take to solve the same problem with 200 numbers, instead? Answer: just a few seconds if it's tractable; and $C\times 2^{200} = (C\times 2^{100})2^{100} = 2^{100}$ seconds if it's not. That's more than a billion trillion years! To paraphrase Senator Dirksen from the great State of Illinois, a trillion years here, a trillion years there, and pretty soon you're talking real time. Exponentialitis is not a pretty condition. Sadly, it afflicts the entire Jurassic menagerie.

The true nature of the ailment eludes us but this much we know: it's genetic. If any one of the Jurassics is tractable, wonder of wonders, all of them are. Better still: a cure for any one of them could easily be used to heal any of the others. Viewed through the tractability lens, the Jurassics are the same T. rex in different brontosaurus' clothings. Heady stuff! The day Alice can split your song library within a few hours will be the day biologists can fold proteins over breakfast, design new drugs by lunch, and eradicate deadly diseases just in time for dinner. The attendant medical revolution will likely make you live the long, jolly life of a giant Galápagos tortoise (life span: 150 years). Alice's discovery would imply the tractability of all the Jurassics (P=NP in computer lingo). Should the computing gods smile upon us, the practical consequences could be huge.

Granted, there would be a few losers: mostly online shoppers and mathematicians. All commercial transactions on the Internet would cease to be secure and e-business would grind to a halt. (More on this gripping drama in the next section.) The math world would take a hit, too: P=NP would prove Andrew Wiles, the conqueror of Fermat's Last Theorem, no more deserving of credit than his referee. Well, not quite. Mathematicians like to assign two purposes to a proof: one is to convince them that something is true; the other is to help them understand *why* something is true. Tractability bears no relevance to the latter. Still, no one wants to see friendly mathematicians swell the ranks of the unemployed as they get replaced by nano iPods, so the consensus has emerged that P is not NP. There are other reasons, too, but that one is the best because it puts computer scientists in a good light. The truth is, no one has a clue.

To be P or not to be P, that is NP's question. A million-dollar question, in fact. That's how much prize money the Clay Mathematics Institute will award Alice if she resolves the tractability of library splitting. (She will also be shipped to Guantánamo by the CIA, but that's a different essay.) Which side of the NP question should we root for? We know the stakes: a

short existence blessed with online shopping (P≠NP); or the interminable, eBay-less life of a giant tortoise (P=NP). Tough call.

P=NP (Or why you won't find the proof on eBay)

An algorithm proving P=NP might conceivably do for technology what the discovery of the wheel did for land transportation. Granted, to discover the wheel is always nice, but to roll logs in the mud has its charms, too. Likewise, the intractability of proof-finding would have its benefits. That 1951 vintage Wham-O hula hoop you bought on eBay the other day, er, you didn't think the auction was secure just because online thieves were too hip for hula hoops, did you? What kept them at bay was the (much hoped-for) intractability of integer factorization.

Say what? Prime numbers deterring crooks? Indeed. Take two primes, S and T, each one, say, a thousand-digit long. The product R= S × T is about 2,000 digits long. Given S and T, your laptop will churn out R in a flash. On the other hand, if you knew only R, how hard would it be for you to retrieve S and T? Hard. Very hard. Very very hard. Repeat this until you believe it because the same algorithm that would find S and T could be used to steal your credit card off the Internet!

Cryptology will help you win wars and shop online

Am I implying that computer security is premised on our inability to do some silly arithmetic fast enough? I surely am. If the Jurassics were shown to be tractable, not a single computer security system would be safe. Which is why for eBay to auction off a proof of P=NP would be suicidal. Worse: factoring is not even known—or, for that matter, thought—to be one of the Jurassics. It could well be a cuddly pet dinosaur eager to please its master (if only its master had the brains to see that). One cannot rule out the existence of a fast factoring algorithm that would have no incidence on the P=NP question.

In fact, such an algorithm exists. All of the recent hoopla about quantum computing owes to the collective panic caused by Peter Shor's discovery that factoring is tractable on a quantum iPod. That building the thing itself is proving quite hopeless has helped to calm the frayed nerves of computer security experts. And yet there remains the spine-chilling possibility that maybe, just maybe, factoring is doable in practice on a

humble laptop. Paranoid security pros might want to hold on to their prozac a while longer.

Cryptology is a two-faced Janus. One side studies how to decrypt the secret messages that bad people exchange among one another. That's cryptanalysis: think Nazi code, Bletchley Park, victory parade, streamers, confetti, sex, booze, and rock 'n' roll. The other branch of the field, cryptography, seeks clever ways of encoding secret messages for good people to send to other good people, so that bad people get denied the streamers, the confetti, and all the rest. Much of computer security relies on *public-key cryptography*. The idea is for, say, eBay to post an encryption algorithm on the web that everybody can use. When you are ready to purchase that hula hoop, you'll type in your credit card information into your computer, encrypt it right there, and then send the resulting gobbledygook over the Internet. Naturally, the folks at eBay will need their own *secret* decryption algorithm to make sense of the junk they'll receive from you. (Whereas poor taste is all you'll need to make sense of the junk you'll receive from them.) The punchline is that no one should be able to decrypt anything unless they have that secret algorithm in their possession.

"Remember, guys, not a word about our factoring algorithm, okay?"

Easier said than done. Consider the fiendishly clever algorithm that encodes the first two words of this sentence as *dpotjefs uif*. So easy to encrypt: just replace each letter in the text by the next one in the alphabet. Now assume you knew this encryption scheme. How in the world would you go about decrypting a message? Ah, this is where algorithmic genius kicks in. (Algorithmistas get paid the big bucks for a reason.) It's a bit technical so I'll write slowly: replace *each* letter in the ciphertext by the *previous* one in the alphabet. Ingenious, no? And fast, too! The only problem with the system is that superior minds can crack it. So is there a cryptographic scheme that is unbreakable, irrespective of how many geniuses roam the earth? It should be child's play to go one way (encrypt) but a gargantuan undertaking to go back (decrypt)—unless, that is, one knows the decryption algorithm, in which case it should be a cinch.

RSA, named after Ron Rivest, Adi Shamir, and Len Adleman, is just such a scheme. It's an exceedingly clever, elegant public-key cryptosystem that, amazingly, requires only multiplication and long division. It rules e-

commerce and pops up in countless security applications. Its universal acclaim got its inventors the Turing award (the “Nobel prize” of computer science). More important, it got Rivest a chance to throw the ceremonial first pitch for the first Red Sox-Yankees game of the 2004 season. Yes, *RSA* is *that* big! There is one catch, though (pun intended): if factoring proves to be tractable then it's bye-bye *RSA*, hello shopping mall.

The computational art of persuasion

Isn't intractability just a variant of undecidability, the mother's milk of logicians? One notion evokes billions of years, the other eternity—what's the difference? Whether the execution of [*program* | *data*] ever terminates is undecidable. In other words, one cannot hope to find out by writing another program and reading the output of [*another program* | [*program* | *data*]]. Of side interest, note how the whole document [*program* | *data*] is now treated as mere data: an artful cadenza from Maestro Turing.

Very nice, but how's undecidability helping us go through life with a smile on our face? It doesn't. In fact, no one ever tried to benefit from an undecidable problem who didn't wind up slouched face down on the Viennese gentleman's couch. Not so with intractable problems. Just as quantum mechanics shattered the platonic view of a reality amenable to noninvasive observation, tractability has clobbered classical notions of identity, randomness, and knowledge. And that's a good thing.

Why? Let me hereby declare two objects to be “identical” if to tell them apart is intractable, *regardless* of how different they might actually be. A deck of cards will be “perfectly” shuffled if it's impossible to prove it otherwise in polynomial time. It is one of the sweet ironies of computing that the existence of an intractable world out there makes our life down here so much easier. Think of it as the Olympics in reverse: if you *can't* run the 100-meter dash under 10 seconds, you win the gold!

Scientists of all stripes are insatiable consumers of random numbers: try taking a poll, conducting clinical trials, or running a lottery without them! To produce randomness can be quite arduous. To this day, only two methods have been scientifically validated. One of them is the infamous “*Kitty Flop*.” Strap buttered toast to the back of a cat and drop the animal from a PETA-approved height: if the butter hits the ground, record a 1; else a 0. For more bits, repeat. Randomness results from the tension between Murphy's law and the feline penchant for landing on one's feet. The other method is the classical “*Coriolis Flush*.” This time, go to the

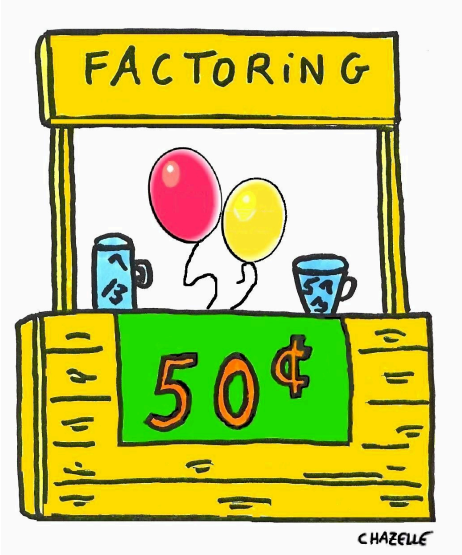
equator and flush the toilet: if the water whirls clockwise, your random bit is a 1; else it's a 0.

Now think how much easier it'd be if cheating were allowed. Not even bad plumbing could stop you (though many hope it would). Okay, your numbers are not truly random and your cards are not properly shuffled, but if to show they are not is intractable then why should you care? Hardness creates easiness. Of course, computer scientists have simply rediscovered what professional cyclists have known for years: the irresistible lure of intractability (of drug detection).

You're not thinking, I hope, that this is all perched on the same moral high ground as Don Corleone's philosophy that crime is not breaking the law but getting caught. If you are, will you please learn to think positive? Our take on intractability is really no different from the 1894 Supreme Court decision in Coffin vs US that introduced to American jurisprudence the maxim "*Innocent until proven guilty*." Reality is not what *is* but what can be *proven to be* (with bounded patience). If you think this sort of tractability-induced relativism takes us down the garden path, think again. It actually cleanses classical notions of serious defects.

Take knowledge, for example: here's something far more faith-based than we'd like to admit. We "know" that the speed of light is constant, but who among us has actually bothered to measure it? We know because we trust. Not all of us have that luxury. Say you're a fugitive from the law. (Yes, I know, your favorite metaphor.) The authorities don't trust you much and —one can safely assume—the feeling is mutual. How then can you convince the police of your innocence? Reveal too little and they won't believe you. Reveal too much and they'll catch you. Intractability holds the key to the answer. And the Feds hold the key to my prison cell if I say more. Sorry, nothing to see here, move along.

Years have passed and you've traded your fugitive's garb for the funky duds of a math genius who's discovered how to factor integers in a flash. Sniffing a business opportunity, you offer to factor anybody's favorite number for a small fee. There might be a huge market for that, but it's less clear there's nearly enough gullibility around for anyone to take you up on your offer—especially with your mugshot still hanging in the post office. No one is likely to cough up any cash unless they can see the

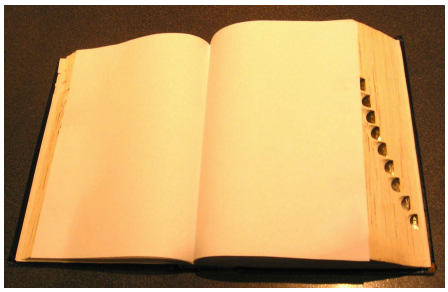


Fresh, juicy primes!

prime factors. But then why would you reward such distrust by revealing the factors in the first place? Obviously, some confidence-building is in order.

What will do the trick is a dialogue between you and the buyer that persuades her that you know the factors, all the while leaking no information about them whatsoever. Amazingly, such an unlikely dialogue exists: for this and, in fact, for any of our Jurassics. Alice can convince you that she can split up your iPod library evenly without dropping the slightest hint about how to do it. (A technical aside: this requires a slightly stronger intractability assumption than P≠NP.) Say hello to the great *zero-knowledge* (*ZK*) paradox: a congenital liar can convince a hardened skeptic that she knows something without revealing a thing about it. *ZK* dialogues leave no option but for liars to tell the truth and for doubting Thomases to believe. They render dishonesty irrelevant, for trusting comes naturally to a society where all liars get caught.

What's intractability got to do with it? Everything. If factoring were known to be tractable, the buyer would need no evidence that you could factor: she could just do it herself and ignore your services—bakers don't buy bread. At this point, the reader might have a nagging suspicion of defective logic: if factoring is so hard, then who's going to be the seller? Superman? In e-commerce applications, numbers to be factored are formed by multiplying huge primes together. In this way, the factors are known ahead of time to those privy to this process and live in intractability limboland for all others.



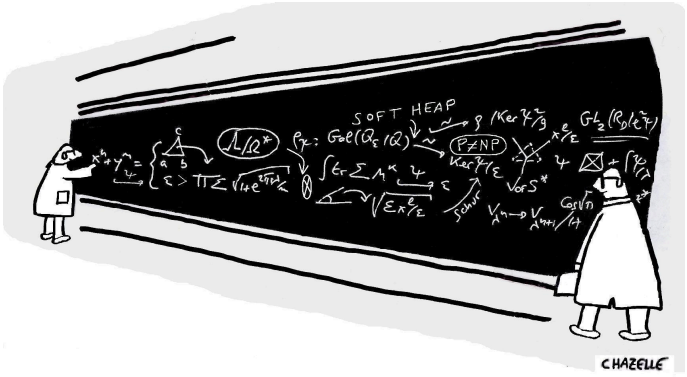
The book of zero-knowledge

It gets better. Not only can two parties convince each other of their respective knowledge without leaking any of it; they can also reason about it. Two businessmen get stuck in an elevator. Naturally, a single thought runs through their minds: finding out who's the wealthier. Thanks to *ZK* theory, they'll be able to do so without revealing anything about their own worth (material worth, that is—the other kind is already in full view).

Feel the pain of two nuclear powers, Learsiland and Aidniland. Not being signatories to the Nuclear Non-Proliferation Treaty, only *they* know the exact size of their nuclear arsenals (at least one hopes they do). Computing theory would allow Learsiland to prove to Aidniland that it outnukes it without leaking any information about its deterrent's strength. The case of Nariland is more complex: it only wishes to demonstrate compliance with the NPT (which it's signed) without

revealing any information about its nuclear facilities. While these questions are still open, they are right up *ZK* 's alley. Game theorists made quite a name for themselves in the Cold War by explaining why the aptly named MAD strategy of nuclear deterrence was not quite as mad as it sounded. Expect *zero-knowledgists* to take up equally daunting "doomsday" challenges in the years ahead. And, when they do, get yourself a large supply of milk and cookies, a copy of Kierkegaard's "Fear and Trembling," and unrestricted access to a deep cave.

More amazing than *ZK* still is this thing called *PCP* (for "probabilistically checkable proofs; *not* for what you think). For a taste of it, consider the sociological oddity that great unsolved math problems seem to attract crackpots like flypaper. Say I am one of them. One day I call the folks over at the Clay Math Institute to inform them that I've just cracked the Riemann hypothesis (the biggest baddest beast in the math jungle). And could they please deposit my million-dollar check into my Nigerian account presto? Being the gracious sort, Landon and Lavinia Clay indulge me with a comforting "*Sure,*" while adding the perfunctory plea: "*As you know, we're a little fussy about the format of our math proofs. So please make sure yours conforms to our standards—instructions available on our web site, blah, blah.*" To my relief, that proves quite easy—even with that damn caps lock key stuck in the down position—and the new proof is barely longer than the old one. Over at Clay headquarters, meanwhile, no one has any illusions about me (fools!) but, bless the lawyers, they're obligated to verify the validity of my proof.



"Gotta run. Let's try PCP ! "

To do that, they've figured out an amazing way, the *PCP* way. It goes like this: Mr and Mrs Clay will pick four characters from my proof at random and throw the rest in the garbage without even looking at it. They will then assemble the characters into a four-letter word and read it out loud very slowly—it's not broadcast on American TV, so it's okay. Finally, based on that word *alone*, they will declare my proof valid or bogus. The kicker: their conclusion will be correct! Granted, there's a tiny chance of error due to the use of random numbers, but by repeating this little game a few times they can make a screwup less likely than having their favorite baboon type all of Hamlet in perfect Mandarin.

At this point, no doubt you're wondering whether to believe this mumbo-jumbo might require not only applying *PCP* but also *smoking* it. If my proof is correct, I can see how running it through the Clays' gauntlet of checks and tests would leave it unscathed. But, based on a lonely four-letter word, how will they know I've cracked Riemann's hypothesis and not a baby cousin, like the Riemann hypothesis for function fields, or a baby cousin's baby cousin like 1+1=2? If my proof is bogus (perish the thought) then their task seems equally hopeless. Presumably, the formatting instructions are meant to smear any bug across the proof so as to corrupt any four letters picked at random. But how can they be sure that, in order to evade their dragnet, I haven't played fast and loose with their silly formatting rules? Crackpots armed with all-caps keyboards will do the darndest thing. Poor Mr and Mrs Clay! They must check not only my math but also my abidance by the rules. So many ways to cheat, so few things to check.

When Abu's light was shining on Baghdad

PCP is the ultimate lie-busting device. Why ultimate? Because it is instantaneous and foolproof. The time-honored approach to truth finding is the court trial, where endless questioning between two parties, each one with good reasons to lie, leads to the truth or to a mistrial, but *never* to an erroneous judgment (yes, I know). *PCP* introduces the instant-trial system. Once the case has been brought before the judge, it is decided on the spot after only a few seconds of cross-examination. Justice is fully served; and yet the judge will go back to her chamber utterly clueless as to what the case was about. *PCP* is one of the most amazing algorithms of our time. It steals philosophy's thunder by turning on its head basic notions of evidence, persuasion, and trust. Somewhere, somehow, Ludwig the Tractatus Man is smiling.

To say that we're nowhere near resolving P vs NP is a safe prophecy. But why? There are few mysteries in life that human stupidity cannot account for, but to blame the P=NP conundrum on the unbearable lightness of our addled brains would be a cop-out. Better to point the finger at the untamed power of the Algorithm—which, despite rumors to the contrary, was *not* named after Al Gore but after Abū ʻAbd Allāh Muhammad ibn Mūsā al-Khwārizmī. As *ZK* and *PCP* demonstrate, tractability reaches far beyond the racetrack where computing competes for speed. It literally forces us to think differently. The agent of change is the ubiquitous Algorithm. Let's look over the horizon where its disruptive force beckons, shall we?

Thinking algorithmically

Algorithms are often compared to recipes. As clichés go, a little shopworn perhaps, but remember: no metaphor that appeals to one's stomach can be truly bad. Furthermore, the literary analogy is spot-on. Algorithms are —and should be understood as—works of literature. The simplest ones are short vignettes that loop through a trivial algebraic calculation to paint *fractals*, those complex, pointillistic pictures much in vogue in the sci-fi movie industry. Just a few lines long, these computing zingers will print the transcendental digits of π, sort huge sets of numbers, model dynamical systems, or tell you on which day of the week your 150th birthday will fall (something whose relevance we've already covered). Zingers can do everything. For the rest, we have, one notch up on the sophistication scale, the sonnets, ballads, and novellas of the algorithmic world. Hiding behind their drab acronyms, of which *RSA*, *FFT*, *SVD*, *LLL*, *AKS*, *KMP*, and *SVM* form but a small sample, these marvels of ingenuity are the engines driving the algorithmic revolution currently underway. (And, yes, you may be forgiven for thinking that a computer geek's idea of culinary heaven is a nice big bowl of alphabet soup.) At the rarefied end of the literary range, we find the lush, complex, multilayered novels. The Algorithmistas' pride and joy, they are the big, glorious tomes on the coffee table that everyone talks about but only the fearless read.

Give it to them, algorithmic zingers know how to make a scientist swoon. No one who's ever tried to calculate the digits of π by hand can remain unmoved at the sight of its decimal expansion flooding a computer screen like lava flowing down a volcano. Less impressive perhaps but just as useful is this deceptively simple data retrieval technique called *binary search*, or *BS* for short. Whenever you look up a friend's name in the phone book, chances are you're using a variant of *BS*—unless you're the patient type who prefers *exhaustive search* (*ES*) and finds joy in combing through the directory alphabetically till luck strikes. Binary search is *exponentially* (ie, incomparably) faster than *ES*. If someone told

“ ‘fetch branch push load store jump fetch… ’ Who writes this crap? ”

you to open the phone book in the middle and check whether the name is in the first or second half; then ordered you to repeat the same operation in the relevant half and go on like that until you spotted your friend's name, you would shoot back: "*That's BS!*"

Well, yes and no. Say your phone book had a million entries and each step took one second: *BS* would take only twenty seconds but *ES* would typically run for five days! Five days?! Imagine that. What if it were an emergency and you had to look up the number for 911? (Yep, there's no low to which this writer won't stoop.) The key to binary search is to have an *ordered* list. To appreciate the relevance of sorting, suppose that you forgot the name of your friend (okay, acquaintance) but you had her number. Since the phone numbers typically appear in quasi-random order, the name could just be anywhere and you'd be stuck with *ES*. There would be two ways for you to get around this: to be the famous Thomas Magnum and bribe the Honolulu police chief to get your hands on the reverse directory; or to use something called a *hash table*: a key idea of computer science.

Hash table? Hmm, I know what you're thinking: Algorithmistas dig *hash* tables; they're down for *PCP*; they *crack* codes; they get bent out of shape by *morphin'*; they swear by *quicksnort* (or whatever it's called). Coincidence? Computer scientists will say yes, but what else are they supposed to say?

Algorithms for searching the phone book or spewing out the digits of π are race horses: their sole function is to run fast and obey their masters. Breeding Triple Crown winners has been high on computer science's agenda—too high, some will say. Blame this on the sheer exhilaration of the sport. Algorithmic racing champs are creatures of dazzling beauty, and a chance to breed them is a rare privilege. That said, whizzing around the track at lightning speed is not the be-all and end-all of algorithmic life. Creating magic tricks is just as highly prized: remember *RSA*, *PCP*, *ZK*. The phenomenal rise of Google's fortunes owes to a single algorithmic gem, *PageRank*, leavened by the investing exuberance of legions of believers. To make sense of the World Wide Web is algorithmic in a qualitative sense. Speed is a secondary issue. And so *PageRank*, itself no slouch on the track, is treasured for its brains, not its legs.

Hold on! To make sense of the world, we have math. Who needs algorithms? It is beyond dispute that the dizzying success of 20th century science is, to a large degree, the triumph of mathematics. A page's worth of math formulas is enough to explain most of the physical phenomena around us: why things fly, fall, float, gravitate, radiate, blow up, etc. As Albert Einstein said, "*The most incomprehensible thing about the universe is that it is comprehensible*." Granted, Einstein's assurance that something is comprehensible might not necessarily reassure everyone,

but all would agree that the universe speaks in one tongue and one tongue only: mathematics.

"***Don't google us, we'll google you.***"

But does it, really? This consensus is being challenged today. As young minds turn to the sciences of the new century with stars in their eyes, they're finding old math wanting. Biologists have by now a pretty good idea of what a cell looks like, but they've had trouble figuring out the magical equations that will explain what it does. How the brain works is a mystery (or sometimes, as in the case of our 43rd president, an overstatement) whose long, dark veil mathematics has failed to lift.

Economists are a refreshingly humble lot—quite a surprise really, considering how little they have to be humble about. Their unfailing predictions are rooted in the holy verities of higher math. True to form, they'll sheepishly admit that this sacred bond comes with the requisite assumption that economic agents, also known as humans, are benighted, robotic dodos—something which unfortunately is not always true, even among economists.

A consensus is emerging that, this time around, throwing more differential equations at the problems won't cut it. Mathematics shines in domains replete with symmetry, regularity, periodicity—things often missing in the life and social sciences. Contrast a crystal structure (grist for algebra's mill) with the World Wide Web (cannon fodder for algorithms). No math formula will ever model whole biological organisms, economies, ecologies, or large, live networks. Will the Algorithm come to the rescue? This is the next great hope. The algorithmic lens on science is full of promise—and pitfalls.

First, the promise. If you squint hard enough, a network of autonomous agents interacting together will begin to look like a giant distributed algorithm in action. Proteins respond to local stimuli to keep your heart pumping, your lungs breathing, and your eyes glued to this essay—how more algorithmic can anything get? The concomitance of local actions and reactions yielding large-scale effects is a characteristic trait of an algorithm. It would be naive to expect mere formulas like those governing the cycles of the moon to explain the cycles of the cell or of the stock market.

Contrarians will voice the objection that an algorithm is just a math formula in disguise, so what's the big hoopla about? The answer is: yes, so

what? The issue here is not logical equivalence but expressibility. Technically, number theory is just a branch of set theory, but no one thinks like that because it's not helpful. Similarly, the algorithmic paradigm is not about *what* but *how* to think. The issue of expressiveness is subtle but crucial: it leads to the key notion of *abstraction* and is worth a few words here (and a few books elsewhere).

Remember the evil Brazilian butterfly? Yes, the one that idles the time away by casting typhoons upon China with the flap of a wing. This is the stuff of legend and tall tales (also known as chaos theory). Simple, zinger-like algorithms model this sort of phenomenon while neatly capturing one of the tenets of computing: the capacity of a local action to unleash colossal forces on a global scale; complexity emerging out of triviality.

Al-Khwarizmi takes wing

Create a virtual aviary of simulated geese and endow each bird with a handful of simple rules: *(1) Spot a flock of geese? Follow its perceived center; (2) Get too close to a goose? Step aside; (3) Get your view blocked by another goose? Move laterally away from it;* etc. Release a hundred of these critters into the (virtual) wild and watch a distributed algorithm come to life, as a flock of graceful geese migrate in perfect formation. Even trivial rules can produce self-organizing systems with patterns of behavior that look almost “intelligent.” Astonishingly, the simplest of algorithms mediate that sort of magic.

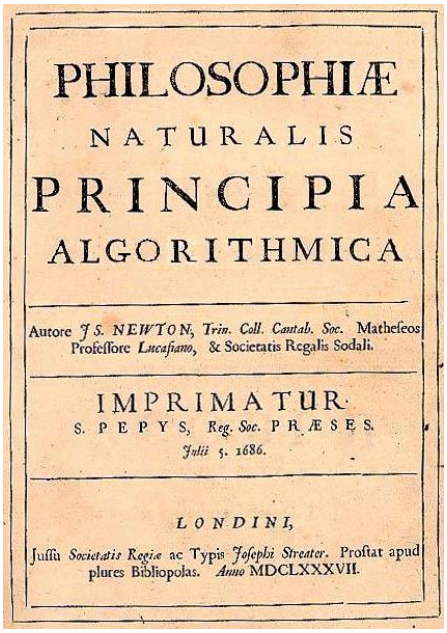
The local rules of trivial zingers carry enough punch to produce complex systems; in fact, by Church-Turing universality, to produce *any* complex system. Obviously, not even algorithmic sonnets, novellas, or Homeric epics can beat that. So why bother with the distinction? Perhaps for the same reason the snobs among us are loath to blur the difference between Jay Leno and Leo Tolstoy. But isn't “War and Peace” just an endless collection of one-liners? Not quite. The subtlety here is called *abstraction*. Train your binoculars on a single (virtual) goose in flight and you'll see a bird-brained, rule-driven robot flying over Dullsville airspace. Zoom out and you'll be treated to a majestic flock of birds flying in formation. Abstraction is the ability to choose the zoom factor. Algorithmic novels allow a plethora of abstraction levels that are entirely alien to zingers.

Take war, for example. At its most basic, war is a soldier valiantly following combat rules on the battlefield. At a higher level of abstraction, it is a clash of warfare strategies. Mindful of Wellington's dictum that Waterloo was won on the playing fields of Eton (where they take their pillow fighting seriously), one might concentrate instead on the schooling of the officer corps. Clausewitz devotees who see war as politics by other means will adjust the zoom lens to focus on the political landscape. Abstraction can be vertical: a young English infantryman within a platoon within a company within a battalion within a regiment within a mass grave on the banks of the Somme.

Or it can be horizontal: heterogeneous units interacting together within an algorithmic "ecology." Unlike zingers, algorithmic novels are complex systems in and of themselves. Whereas most of what a zinger does contributes directly to its output, the epics of the algorithmic world devote most of their energies to servicing their constituent parts via swarms of intricate *data structures*. Most of these typically serve functions that bear no direct relevance to the algorithm's overall purpose—just as the mRNA of a computer programmer rarely concerns itself with the faster production of Java code.

The parallel with biological organisms is compelling but far from understood. To this day, for example, genetics remains the art of writing the captions for a giant cartoon strip. Molecular snapshots segue from one scene to the next through plots narrated by circuit-like chemical constructs—zingers, really—that embody only the most rudimentary notions of abstraction. Self-reference is associated mostly with self-replication. In the algorithmic world, by contrast, it is the engine powering the complex recursive designs that give abstraction its amazing richness: it is, in fact, the very essence of computing. Should even a fraction of that power be harnessed for modeling purposes in systems biology, neuroscience, economics, or behavioral ecology, there's no telling what might happen (admittedly, always a safe thing to say). To borrow the Kuhn cliché, algorithmic thinking could well cause a paradigm shift. Whether the paradigm shifts, shuffles, sashays, or boogies its way into the sciences, it seems destined to make a lasting imprint.

Now the pitfalls. What could disrupt the rosy scenario we so joyfully scripted? The future of the Algorithm as a modeling device is not in doubt. For its revolutionary impact to be felt in full, however, something else needs to


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
ALGORITHMICA
Autore J.S. NEWTON, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, & Societatis Regalis Sodali.
IMPRIMATUR
S. PEPYS, Reg. Soc. PRÆSES.
Julii 5. 1686.
LONDINI,
Jussu Societatis Regiæ ac Typis Josephi Streater. Prostat apud plures Bibliopolas. Anno MDCLXXXVII.

Had Newton been hit by a flying goose and not a falling apple...

happen. Let's try a thought experiment, shall we? You're the unreconstructed Algorithm skeptic. Fresh from splitting your playlist, Alice, naturally, is the advocate. One day, she comes to you with a twinkle in her eye and a question on her mind: "*What are the benefits of the central law of mechanics?*" After a quick trip to Wikipedia to reactivate your high school physics neurons and dust off the cobwebs around them, you reply that *F=ma* does a decent job of modeling the motion of an apple as it is about to crash on Newton's head: "*What's not to like about that?*" "*Oh, nothing,*" retorts Alice, "*except that algorithms can be faithful modelers, too; they're great for conducting simulations and making predictions.*" Pouncing for the kill, she adds: "*By the way, to be of any use, your vaunted formulas will first need to be converted into algorithms.*" Touché.

Ahead on points, Alice's position will dramatically unravel the minute you remind her that *F=ma* lives in the world of calculus, which means that the full power of analysis and algebra can be brought to bear. From *F=ma*, for example, one finds that: (i) the force doubles when the mass does; (ii) courtesy of the law of gravity, the apple's position is a quadratic function of time; (iii) the invariance of Maxwell's equations under constant motion kills *F=ma* and begets the theory of special relativity. And all of this is done with *math alone!* Wish Alice good luck trying to get her beloved algorithms to pull that kind of stunt. Math gives us the tools for doing physics; more important, it gives us the tools for doing math. We get not only the equations but also the tools for modifying, combining, harmonizing, generalizing them; in short, for reasoning about them. We get the characters of the drama as well as the whole script!

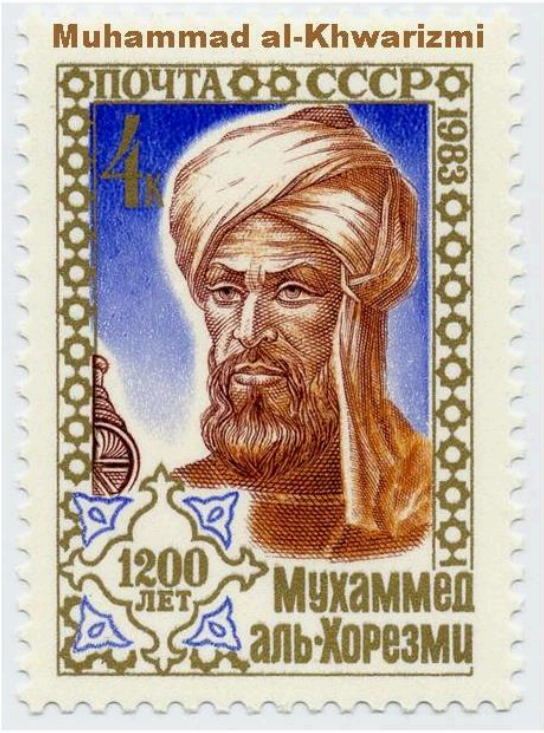
Is there any hope for a "calculus" of algorithms that would enable us to knead them like Play-Doh to form new algorithmic shapes from old ones? Algebraic geometry tells us what happens when we throw in a bunch of polynomial equations together. What theory will tell us what happens when we throw in a bunch of algorithms together? As long as they remain isolated, free-floating creatures, hatched on individual whims for the sole purpose of dispatching the next quacking duck flailing in the open-

problems covey, algorithms will be parts without a whole; and the promise of the Algorithm will remain a promise deferred.

While the magic of algorithms has long held computing theorists in its thrall, their potential power has been chronically underestimated; it's been the life story of the field, in fact, that they are found to do one day what no one thought them capable of doing the day before. If proving limitations on algorithms has been so hard, maybe it's because they can do so much. Algorithmistas will likely need their own “Google Earth” to navigate the treacherous canyons of Turingstan and find their way to the lush oases amid the wilderness. But mark my words: the algorithmic land will prove as fertile as the one the Pilgrims found in New England and its settlement as revolutionary.

Truth be told, the 1776 of computing is not quite upon us. If the Algorithm is the New World, we are still building the landing dock at Plymouth Rock. Until we chart out the vast expanses of the algorithmic frontier, the P vs NP mystery is likely to remain just that. Only when the Algorithm becomes not just a *body* but a *way* of thinking, the young sciences of the new century will cease to be the hapless nails that the hammer of old math keeps hitting with maniacal glee.

One thing is certain. Moore's Law has put computing on the map: the Algorithm will now unleash its true potential. That's one prediction Lord Kelvin never made, so you may safely trust the future to be kind to it.



“May the Algorithm's Force be with you.”