International Journal of Theoretical Physics, Vol. 42, No. 2, February 2003 (© 2003)

Debugging the Universe[1]

Brian Hayes[2]

Many authors have asked whether our universe can be understood as a kind of computational process. This paper asks the opposite question: What are the challenges we would face in writing a program (or, more generally, creating a computation) to implement a physically plausible universe?

KEY WORDS: computer simulation; computational metaphysics; N-body problem; cellular automata.

1. INTRODUCTION

The first great age of automata began at the close of the medieval period and lasted into the Seventeenth century (Mayr, 1986). The technological marvels of that era were clockwork confections machined from brass and iron—intricate assemblies of gears, cranks, levers, escapements, and ratchets. The clocks of cathedrals and town halls displayed the phases of the moon and the progress of the sun through the zodiac; animated figures pranced out to strike the hours and entertain onlookers (King, 1987; Lloyd, 1958; Maurice and Mayr, 1980). There was even a mechanized doll that took up a quill pen and wrote, "I think, therefore I am" (Wood, 2002).

From machines that imitate life and the heavens, it is an easy step to the idea that life itself might be a mechanical process and that the stars could be driven by some kind of celestial gear train. The vision of a clockwork universe figures in the thinking of Dante, Galileo, Kepler, and Newton. Yet another exponent of gears in the sky was Descartes, who also likened animals to mechanical automata. Then there was Rousseau, who wrote: "I see nothing in any animal but an ingenious machine, to which nature hath given senses to wind itself up . . . ." Ideas like these

[1] This paper is loosely based on a talk given at the Digital Perspectives meeting in Arlington, VA, in July 2001.

[2] Senior Writer, *American Scientist*; 99 Alexander Drive Research Triangle Park, NC, 27709. USA; e-mail: bhayes@amsci.org.

0020-7748/03/0200-0277/0 © 2003 Plenum Publishing Corporation

were very much in the air a few hundred years ago; and yet, as far as I can tell, the clockwork universe and the animal-as-automaton were always looked upon as metaphors rather than literal mechanisms. The universe was *like* a clock, but no one believed that the meshing of brass gears really moved the planets in their orbits. And Rousseau knew that if he were to open up a living creature, he would not find a mainspring inside.

Today we live in another age of automata, though ours are mostly made of silicon rather than brass. The computer is everywhere in our lives, and so we are predisposed to see the world in terms of computational processes. In particular, if we seek a mechanistic understanding of the natural universe, we are quick to adopt the vocabulary of symbolic logic, algorithms, and information theory, taking the viewpoint that the universe evolves by continually computing its next state. But is this computational model of the universe just another metaphor—another exercise in "as if" reasoning—or should it be interpreted more concretely? Some proponents of the computational universe seem to be fully in earnest when they argue that the ultimate basis of all physics is algorithmic (Fredkin, 1990, 1992, 2002). Others are hesitant or coy on this point, leaving open the possibility that the whole notion is merely a useful fiction, a manner of speaking or thinking (Ilachinski, 2001; Minsky, 1982; Noyes et al., 2001; Schmidhuber, 1997; Wolfram, 2002; Zuse, 1982). And there is a third faction that says the idea is not merely wrong but impossible (Dreyfus, 1972; Penrose, 1989).

Popular culture has added another layer of confusion to this much-muddied subject. For example, three recent films depict characters who discover that their world is some kind of computer-generated simulation, or who step into such a simulation and then have a hard time finding their way out (eXistenZ; The Matrix; The Thirteenth Floor). If stories of this kind are representative of what is meant by the term "computational universe," then the subject may well lie beyond the reach of scientific discourse. Asking whether we are bit players in someone else's virtual reality—or someone else's screen saver—is no more fruitful than asking if we are figments of the Red King's dream (Dodgson, 1960). The question is metaphysical. (On the other hand, Hans Moravec tells a fable of creatures inside a computational world who *do* make the discovery that they live inside a program. They somehow escape this predicament and become colleagues of their creator. (Moravec, 1988))

Let us set aside for now these more flamboyant visions of a computer-generated world. To suppose that the universe is a computational process does not have to imply an intelligent design, or a computation performed for some definite purpose, or the existence of a master programmer who pressed "Enter" on a cosmic keyboard to set the whole works in motion. All that is required is that the ultimate laws of nature have an essentially algorithmic character. If the universe is computational, then at some level all events can be described in terms of a finite set of elementary, deterministic operations suitable for execution by a Turing machine or some other automation. The process of discovering these computational laws of nature—the instruction set of the computational universe—becomes the primary aim of physics, which is thereby transformed into a kind of reverse engineering.

Reverse engineering is a formidable challenge. In the present context, it means taking the output of a computation and inferring or reconstructing the program that generated the output. This inductive process is particularly difficult when you don't know whether a solution exists (the output might not be the product of any computation at all) or whether the solution is unique (many programs might have the same output). To cite the most obvious instance of the latter problem, the astrophysical "programs" suggested by Ptolemy, Copernicus, Kepler, Newton, and Einstein embody very different algorithms, but they were all meant to explain essentially the same data.

The obvious alternative to reverse engineering is forward engineering: Instead of trying to infer the internal structure of nature's program by examining its output, we can write a program of our own, aiming to produce a world something like the natural one. Forward engineering is no cinch either. The task amounts to nothing less than creating a universe—writing a computer program whose output is the physics of some believable world. But this synthetic approach has one key advantage: When we build a computational world of our own, we know in advance, beyond all doubt, that the physics of that world is computable down to the last detail. There can be no spooky indeterminism or supernatural mental intuitions in this created world (unless we put them there, in which case they won't seem very spooky or supernatural). And if something puzzling or unexpected does turn up in the model universe, we can track it down in the source code—we can debug the universe. In sum: A universe we make is a universe we can *know*.

There is another advantage to the synthetic approach. If we try to imagine the details of a program that might be running our own universe, there is no avoiding some rather difficult physics. For example, any computational universe inconsistent with relativistic quantum field theories would be a nonstarter. But when we set out to build worlds *de novo,* we can make them as simple as we wish. In particular, we can build a purely classical universe, based on the physics of Newton and Laplace—or on the physics of Aristotle, for that matter. In most of what follows I shall assume that the computed world is indeed a simple place.

2. GROUND RULES

In a sense, the trouble with building a do-it-yourself universe is that it's too easy. Playing in your own sandbox, you can get away with almost anything; you can make up whatever laws of nature please your fancy. Criteria for judging the success of such a project tend to be disappointingly subjective and aesthetic. There are too few constraints.

These criticisms are serious ones, and need to be borne in mind. But it is not quite true that we can do anything we want when we invent a computational universe. The computational process itself imposes a few constraints. Assuming that the programmed universe is to run on a machine something like the ones we know today, here are two rules that cannot be violated:

- *No continuum.* Because we have only finite computational resources available, all quantities in the model universe must be of finite magnitude, and they can be computed with only finite precision, or else selected from finite sets. Of particular importance, space and time must have some kind of discrete structure, so that coordinates can always be expressed with no more than a finite amount of information. Philosophers may continue to debate the status of the continuum in our own world, but for any universe that fits inside a computer we know how to build, the issue is settled.
- *No randomness.* Because the logical structure of our computers is strictly deterministic, so are all the events that take place in the computed universe. The chain of causality may be arbitrarily tangled, but in principle it can always be traced. The only way around this restriction would be to import randomness into the model from our own world—borrowing it from some source that we do not fully understand ourselves. For present purposes I regard this as a form of cheating (since the universe created is not entirely specified by the program), and I assume it will not be done.

Common sense suggests a few more constraints, which we may choose to impose on ourselves. There is nothing mandatory about these rules—it's easy to imagine a computational universe that would not observe them—but without them the game of computational cosmogony is not very interesting.

- *No miracles.* Like the original clockwork universe, this one is to be set ticking and then left to evolve on its own. The programmer is not allowed to intervene in its operation. A stronger version of this rule insists that only the simplest and most elementary events are to be specified directly in the model; everything else must be an emergent phenomenon. For example, the model might spell out how elementary particles interact to build nuclei and atoms, but the agglomeration of the atoms to form molecules and larger structures would not be separately specified.
- *No shortcuts.* Any approximations or simplifications built into the program become exact laws of nature in the computed world. So do any numerical roundoff errors. If calculations are done with IEEE floating-point arithmetic, then scientists in the computed universe will find that nothing can be measured with more than 16 digits of precision. The no-shortcuts rule has an interesting interaction with the no-continuum rule: In general we think of $\Delta x/\Delta t$ as an approximation to dx/dt, which becomes exact only in the limit where Δx and Δt go to zero. But in this context there must be some nonzero Δx and Δt that give exact results. Differential equations

become approximations to finite-difference equations, rather than the other way around.

- *No peeking.* The state of the world at time t can depend only on events at times prior to t. The program will not look into the future and use what it learns there to reconstruct the past. There are arguments for imposing an even stronger condition: At each instant t, the program must calculate the state of the world at time $t+1$ based solely on the state at t. "Leapfrogging" of information from $t-1$ to $t+1$ is forbidden. The point of these restrictions is to enforce a mode of computation in which the state of the world evolves or unfolds, one instant at a time, just as we perceive our own existence. Note that the no-peeking rule has nothing to say about time-reversal symmetry *within* the computed universe. Microscopic events, such as particle collisions, might well be reversible; but the program computing these events cannot jump forward and backward in time.

The kind of program envisioned here is not a *simulation* of physics but an *implementation* of it. The distinction has to do with both method and intent. For example, a simulation of the dynamics of matter in a galaxy might well operate in a "collisionless" regime, where stars interact through long-range forces but never touch one another—even if they happen to pass through the same point of space at the same instant. As long as the density of stars is not too high, this simplification has little effect, and the simulation can still make quite accurate predictions. But the idea behind the computational universe is not just to make correct predictions but to identify correct mechanisms. We want verisimilitude as well as accuracy. It's not enough that the output of a program mimic natural phenomena. We should also be able to look inside the program, at the underlying algorithms and data structures, and say, "Yes, nature could do it that way too."

In what follows I discuss two broad approaches to building a computational universe, one scheme based on the dynamics of elementary particles and the other on the wavelike mechanics of cellular automata. I compare these computational models primarily by asking questions about the programs themselves, not about the universes they generate. In other words, I am evaluating them according to the criteria of computer science, not those of physics.

3. PARTICLE MECHANICS

The conceptual framework that has dominated physics for the past 150 years or more suggests that the most fundamental entities are pointlike particles, which interact with one another by means of forces or fields. This notion is so deeply entrenched in modern consciousness that you need a pretty good reason before you set it aside and choose some other way of thinking about the world. Thus particles and fields are probably the most obvious candidates for basic building blocks of a computational universe.

The simplest version of the particle-and-field universe is purely Newtonian. A particle is a dimensionless object with an exactly defined position and velocity and a few other invariant properties such as mass and electric charge; once we have catalogued these features, there is nothing more to be said about the particle. In other words, a particle can be fully described in a data structure something like this:

```
particle
   has
      rest-mass (constant of type rational)
      electric-charge (constant of type rational)
      spin (constant of type rational)
      x-position (variable of type rational)
      y-position (variable of type rational)
      z-position (variable of type rational)
      x-velocity (variable of type rational)
      y-velocity (variable of type rational)
      z-velocity (variable of type rational)
```

It is worth emphasizing what's *not* present in this data structure. The particle knows its current coordinates and velocity, but not any higher derivatives of these dynamical variables, and it carries around no explicit record of its past states.

If we ignore all interactions between particles, then a very simple program will suffice to run a universe of this kind. At each time step, the program has to update the position and velocity components of each particle; in fact only the positions can chage, because the absence of interactions means there are no accelerations. For a universe with *n* particles, such a program has running time proportional to *n* on a sequential computer. In a parallel machine with *n* processors, the running time is constant (regardless of the computer's architecture; there is no communications overhead). This parallel implementation is particularly appealing if we choose to imagine that the particles themselves are the processors, continually calculating their own trajectories; then the number of processors is automatically matched to the computing load, even if particles are created or annihilated from time to time.

On the other hand, although the computational model is well behaved, the universe it describes is a very dull place. Nothing ever happens there. The particles sail gracefully on their geodesic paths, oblivious of each other's presence. If we want to see any action, we need *interactions*.

Interactions can be added to the model in any of several ways, such as by introducing direct pairwise forces, or by having the particles generate and respond to fields. But first consider an even simpler scheme: a universe of billiard-ball particles, which interact with one another only when they come into direct, tangible contact. Thus the physics of interactions reduces to detecting collisions and predicting the geometry of the rebound paths (which are always geodesics between collisions).

In the world we live in, collision detection doesn't seem like much of a challenge. You don't need to do any computing at all to know when you've stubbed your toe. Likewise, real billiard balls require no instrumentation or intelligence to detect each other's presence; they simply obey the "law of nature" that says two solid bodies cannot occupy the same space at the same time. Ideally, objects in the computational universe would behave in the same way—they would be endowed with the property of solidity, and thus they would automatically rebound from obstacles—but such autonomy of action is not to be found in worlds of our creation. When you build a computational universe, nothing falls to earth unless you remember to turn the gravity on. Nothing bounces unless you tell it exactly where and when to bounce. Translating this principle into the idiom of computer programming, collisions do not generate an interrupt; you have to poll to detect them.

For dimensionless particles moving through a discrete space-time, it's not even entirely clear how best to define a collision. One choice is to say that two particles collide when they lie at the same point in the lattice of spacetime coordinates. Then detecting a collision between particles u and v is just an equality test on the coordinates:

$$(u_x = v_x) \wedge (u_y = v_y) \wedge (u_z = v_z)$$

This scheme has the virtue of simplicity, but it offends against the intuition that two solid bodies cannot be at the same place at the same time. The alternative is to consider two particles as having collided whenever they reach adjacent sites in the lattice. (In statistical mechanics this is known as a hard-core lattice gas.) Here the collision–detection routine becomes somewhat more convoluted. On a cubic lattice, the particles have to evaluate an expression that looks something like this:

$$\begin{gathered}((u_x = v_x) \wedge (u_y = v_y) \wedge ((u_z = v_z - 1) \vee (u_z = v_z + 1))) \vee \\ ((u_x = v_x) \wedge ((u_y = v_y - 1) \vee (u_y = v_y + 1)) \wedge (u_z = v_z)) \vee \\ (((u_x = v_x - 1) \vee (u_x = v_x + 1)) \wedge (u_y = v_y) \wedge (u_x = v_z))\end{gathered}$$

Even if the collision-detection procedure can be made trivially easy, adding it to the programmed universe has grave effects on the overall computational complexity of the process. For a universe with n particles, detecting all collisions requires $n(n-1)/2$ operations at each time step; in other words, the algorithm has

$\mathcal{O}(n^2)$ complexity on a sequential computer.[3] Moreover, providing one processor per particle does not reduce the computation to constant running time. For that we would need one processor for each *pairing* of particles, which is not a concept that maps neatly into any obvious computer architecture. Because of this quadratic complexity, a universe with collision detection will run faster or slower as particles are created or annihilated. Just think: Somewhere in the computed universe there's a burst of matter–antimatter creation, and here in our world the dynamos groan and the lights dim as the machine takes up the added computational load.

Of course a universe full of hard-core-repulsive billiard balls is probably not the model we most want to study anyway. Of greater interest are theories where particles exert long-range attractions or repulsions on one another. But introducing those forces certainly doesn't make the computation any easier. In the naïve algorithm for this n-body problem—based on pairwise interactions—the quadratic complexity remains: Every particle has to consider forces generated by every other particle. The naïve algorithm is not the only possible choice here; there are many other approaches (Blelloch and Narlikar, 1997, Graps, 2000). Most mesh and tree methods (Barnes and Hut, 1986) have $\mathcal{O}(n \log n)$ complexity, and the fast multipole method (Greengard and Rokhlin, 1987) is usually described as $\mathcal{O}(n)$. (But see (Aluru, 1996) for a dissenting view.) However, attaining these speedups entails some level of approximation, since the algorithms lump together certain groups of particles and count only their smoothed or averaged influence. Thus a universe that is supposed to have an inverse-square law of gravitation might exhibit slightly different behavior when force calculations are performed using one of these algorithms. Furthermore, the departures from an inverse-square law would depend on the detailed configuration of the particles. In some cases, the running time would also vary with the mass distribution, so that, for example, a universe with dense clusters would go slower than one with matter spread out more uniformly.[4]

One entirely valid response to all these concerns over algorithmic complexity is a shrug and a sigh. Who cares how fast the universe runs? There is no fundamental reason that the program's execution time could not be quadratic in n, or even exponential in n. Any civilization that evolved within the artificial universe would

[3] An alternative to *detecting* collisions is *solving* for them. Since the particles move in straight lines at constant velocity, all collisions can be predicted by solving a system of linear equations. But this procedure is a flagrant violation of the no-peeking rule. Also, the technique may be less efficient than it seems. After solving for all pairwise collisions, you have to throw away all but the first, since that collision can alter all subsequent trajectories.

[4] Things could be worse. The $\mathcal{O}(n^2)$ running time of a classical n-body algorithm seems quick indeed compared with the equivalent computation in a quantum-mechanical world. Almost all exact quantum-mechanical calculations have exponential running time (when performed on a classical computer). If the wave function of a single particle is specified by m variables, then n particles require not mn variables but m^n. (Ceperley, 1999)

never know anything about the rate of its own computation, because the ticking of clocks inside the program must always remain synchronized with the execution of the program itself. (We can't measure the "speed of time" in our universe either.) Fluctuations in the pace of the algorithm would also be undetectable from the inside. For that matter, the program could be stopped altogether—keeping the created world hovering in suspended animation—and then resumed. No one inside would be any the wiser.

The real objection to quadratic complexity is not an issue of speed or efficiency. Rather, the quadratic running time is a clue that the program is doing something "unnatural," something that we don't see going on in our own world and that we'd rather not include in an artificial physics. A program that monitors all pairs of particles for collisions or that repeatedly measures all pairwise forces probably has somewhere inside it a big list of all the particles in the universe; the calculation is essentially an iteration over this list. In the natural world we find no counterpart of such a data structure. The existence of this master list seems just as implausible and incongruous as polished brass gears that drive the heavenly spheres.

What seems most suspect about the particle-mechanics algorithms is their extreme nonlocality. In these programs, a billiard ball rolling slowly across a green felt table has to be continually checking the positions of all the other billiard balls in the universe, lest a collision go undetected. Perhaps this computation will produce results indistinguishable from the events we observe on real billiard tables, but it also produces an overwhelming sense that there ought to be an easier way.

Locality is an even more contentious issue in physics than it is in computer science, and perhaps a distrust of $\mathcal{O}(n^2)$ algorithms in the context of a computational universe simply mirrors the long debate in physics over action at a distance. For a time, quantum field theory seemed to offer a formalism in which all interactions could be understood in terms of strictly local events, with forces being conveyed between particles by the exchange of other particles. A relativistic quantum field theory might also get rid of the fixed frame of reference implied in the classical computational model of particle mechanics. But there is a steep price to pay for these improvements: Calculating a single exact interaction between particles could entail summing an infinite series of Feynman diagrams. Furthermore, with the constraints imposed by Bell's inequalities, it's not at all clear that even quantum field theories can be made strictly local.

4. WAVE MECHANICS

One way to avoid awkward problems of nonlocality in a computational universe is to change the locus of computation. Instead of associating processors with particles, and having each particle ask all the other particles, "Where are you?," each site in space becomes a processor, and it asks the neighboring sites, "Anybody there?" Thus only adjacent sites have to communicate with one another, and the computation is pleasingly local—or so it seems on first glance. I refer to this computational scheme as "wave mechanics" because of the way information propagates from site to site, as if in a discretized wave.

The obvious implementation of this idea is a cellular automaton—an array of discrete cells communicating with their neighbors and continually executing some rule or program to update their own state. Systems of this kind were first studied in detail around 1950 by John von Neumann and Stanislaw Ulam (vonNeumann and Burks, 1966); Konrad Zuse explored similar themes at about the same time (Zuse, 1982). Two decades later, John Horton Conway's Game of Life popularized the concept of cellular automata and gave intriguing evidence of intricate behavior in these systems (Gardner, 1984). Cellular automata were shown to be an effective tool for modeling and simulation in the physical sciences by the Information Mechanics Group at M.I.T., including Edward Fredkin, Norman Margolus, Tommaso Toffoli and Gérard Y. Vichniac (Fredkin and Toffoli, 1982; Margolus, 1984; Toffoli, 1982; Tofolli and Margolus, 1987; Vichniac, 1984). Toffoli also made the important discovery that interesting dynamics can emerge in cellular automata based on invertible update rules. Later, Stephen Wolfram systematically explored and classified some of the simplest families of cellular automata, namely those defined on a one-dimensional lattice, and showed that complex behavior can be observed even in these systems (Wolfram, 2002).

Cellular automata resemble certain models developed independently in other areas of physics, such as lattice gases in fluid mechanics, the Ising model and its many relatives in condensed-matter physics, and the lattice-gauge-theory formulation of quantum field theories. There are also close connections with the study of symbolic dynamics in mathematics (Lind and Marcus, 1995). The versatility of the idea underlying all these systems is not surprising, because (as Toffoli has emphasized) cellular automata can be viewed as a discrete implementation of differential equations. This also makes them popular candidates for the substrate of a computational universe.

In the simplest cases, a cellular automation is built on a geometrically regular lattice, such as an orthogonal grid, and all the cells are identical: They have the same finite set of possible states, they communicate with the same fixed set of neighbors, and they execute the same program to calculate their next state. The specification of a single cell in such an array might look like this:

```
cell
  has
    state (variable of type integer)
    neighbors (constant array of type cell)
    update-rule (constant of type procedure)
```

Note that once you have chosen the topology of the neighborhood and the number of states per cell, there are only finitely many update rules. Specifically, if a cell with m possible states considers the states of k neighboring cells when calculating its next state, then there are m^{m^k} update rules. This is a large number even for modest values of m and k, but still it is finite. In a sense, then, when we build a cellular automaton we are not designing a universe but simply choosing one from a list of available models.[5]

Ideally, there would be a simple and direct mapping from the elements of the cellular automaton to those of the computed universe. Indeed, the temptation is almost irresistable to identify the cells of the automaton with discrete volumes of space in the universe. In other words, each cell corresponds to a specific point or region in space, and the topological neighbors of the cell are the physical neighbors of the corresponding region. This vision goes beyond "programmable matter"(Toffoli and Margolus, 1991) to "programmable space." Space itself becomes a computational medium, and the universe computes its own evolution, based entirely on the repeated evaluation of local update rules. Conservation laws for basic quantities such as energy and momentum can be built into the operation of the automaton, and there is also a natural analog of the speed of light, since information cannot propagate through the lattice any faster than one cell per time step. With a judicious choice of update rule and neighborhood, one might hope to see interesting behavior emerge at a scale somewhat larger than that of the underlying cells. For example, there might be long-lived, coherent patterns of excitation resembling waves or particles.

Over the years, a variety of cellular automata have been proposed as models of fundamental physics in our universe—and have been met with equally varied criticisms. One frequent objection is that the finite symmetries of the underlying lattice would show through in macroscopic observations, whereas our actual world appears to be isotropic to high precision (and special relativity implies *exact* isotropy). Those who argue for cellular automata as realistic models of physics must answer these criticisms (and indeed responses have been offered), but the issue is of little relevance here, where we are constructing a computational universe of our own design. If the regularities of the lattice show through in the fabric of such a universe, so be it.

Instead of focusing on the physics of the cellular-automaton universe, I want to consider a few aspects of its computation or implementation. One immediate observation is that issues of time complexity are less troublesome here than in a universe based on particle dynamics. Whereas the time needed to update the positions and velocities of interacting particles is a nonlinear function of the number of particles, the computational load in a cellular automaton should be directly

[5] Of course the same argument applies to any system with a finite number of components interacting in finitely many ways, but the countability is seldom so clear as it is in cellular automata.

proportional to the number of cells (and perhaps also to the size of the neighborhood). Given the obvious parallel implementation, with one processor per cell, the running time per update should be a constant, regardless of the number of cells. Furthermore, the running time is independent of the level of activity in the universe: There is no worry that the computation will bog down whenever a new galaxy coalesces from primordial chaos. Unfortunately, the program's constant running time is achieved by choosing always to do the maximum amount of computing rather than the minimum: The cellular automaton does just as much work calculating the next state of a completely vacant universe as one teeming with activity. In the case of a universe like our own, most of the cells would have nothing much to do most of the time. Fredkin calls this "the problem of the missing workload"; by his estimate, the capacity of a cellular automaton computer for our universe is greater than needed by a factor of 10^{63}. "Either something else is going on . . . ," he comments, or "God was incompetent on a scale that boggles the mind." (Fredkin, 1992) Still, from the standpoint of computational complexity theory, this extravagant "waste" of resources is of no consequence; it contributes only a constant factor to the running time, and by the conventions of complexity theory any constant factor, even 10^{63}, reduces to $\mathcal{O}(1)$.

A different kind of timing issue—having to do with the synchronization or sequencing of events in the cellular automaton—is more problematic. To see the source of this difficulty, it's necessary to go into some detail about the internal operation of the automaton. Suppose that each cell displays its current state by raising one of m differently colored flags on a mast. Inside the cell is an algorithmic daemon whose endlessly repeated task is to check the colors of k neighboring flags (possibly including its own flag) and to hoist the appropriate flag on its own mast. The daemon chooses which flag to display by consulting a table that lists all m^k possible combinations of neighboring flags, specifying a state for each such combination. This lookup table constitutes the cell's update function. There are no other inputs to the function apart from the current colors of the k flags in the neighborhood; also, the daemon has no scratchpad for storing and recalling former states of its own cell or of its neighbors. The only element of the system that might correspond to the concept of memory is the state of the cell itself, as visibly encoded in the color of the flag flying from the mast.

This model seems to be fully explicit and deterministic—and for describing the behavior of a single cell, perhaps it is. But when you try to apply it to an array of many cells, something is missing: There is no representation of time. A daemon has to do everything at once—read the flags of the neighboring cells, consult the lookup table, raise its own flag. Meanwhile other daemons are simultaneously checking their neighborhoods and changing their own flags accordingly, which means the first daemon may need to make another change, and so on. In the terminology of circuit engineering this is a *race condition,* and the outcome is indeterminate. (Or, more precisely, the outcome is determined by details of implementation we don't want to think about—which transistor has a slightly higher β or a slightly lower load resistance, etc. These are not the kinds of factors that ought to determine the fate of a universe.)

The quick cure for this disease is to discretize time.[6] We can synchronize all the daemons by adding a global clock signal, perhaps a bell that rings to tell them when to begin their observations. But even this device will not quite solve the problem if the daemons work at different speeds. When the bell rings at time t, the fastest daemon will immediately survey its k neighbors—which are necessarily still displaying signals from time $t-1$—and quickly raise its time-t flag. Hence the neighbors of this cell will see one time-t flag and $k-1$ flags left over from time $t-1$. Daemons elsewhere will observe various other combinations of flags, possibly including indeterminate states in which a cell is displaying two flags at once or no flag at all. Again, the outcome of the process depends on unspecified variables in the implementation of the daemon. What's needed to restore uniformity and determinism is a *two-phase* clock: a bell that prompts all the daemons to make their observations and begin the calculation of the next state, followed some time later by a whistle, which is the signal to raise the flag marking the new state. But in order to implement this scheme, still more is needed: Each cell must now have some way of remembering, during the interval between the ringing of the bell and the blowing of the whistle, which flag it will eventually hoist. In other words, the state of the cell now encompasses more than just the manifest color of the flag on the mast; there are also secret variables,[7] which affect the behavior of the system but are not externally observable.

This scheme for synchronizing the daemons adds a fair amount of complexity to the cellular automaton, and yet it still does not solve all timing problems. In 1984, Vichniac (Vichniac, 1984) analyzed a version of the two-dimensional Ising model implemented as a cellular automaton. In this model each cell has two possible states, interpreted as *up* and *down* spins of the atoms in a ferromagnet. The neighborhood consists of the cells at the four cardinal compass points, and the update rule favors configurations in which adjacent cells have the same state (that is, the spins point in the same direction, as in a magnetized material). With parallel updating as described above—so that a cell choosing a state for time t sees all four neighbors at time $t-1$—Vichniac observed that the system evolves not toward the expected low-energy ground state (with uniform spins, either all *up* or all *down*) but instead toward a physically implausible oscillation, alternating between the two highest-energy states of the lattice, with checkerboard configurations of opposite

[6] By most definitions, the system does not even become a proper cellular automaton until time is made discrete.

[7] A more natural term might be "hidden variables," but this is misleadingly evocative of the controversy over hidden-variable theories in quantum mechanics.

up and *down* spins. The cause of this anomaly is a runaway feedback loop. In the checkerboard geometry, each *up* cell is surrounded by four *down* cells, and vice versa. Accordingly, the update rule causes all the spins to flip on every time step. An *up* spin changes to *down* in order to match its neighbors, but meanwhile all those neighbors have flipped to *up*.

This blinking-checkerboard pathology is certainly not to be seen as a fatal flaw of all cellular automata. It is peculiar to the specific update rule of the Ising model, and even in that case it can be remedied in various ways (such as by employing sequential instead of parallel update). What is unsettling about this situation is the way details of the implementation have leaked into the physics. The original statement of the Ising model says nothing about the order in which spins interact with one another, and one would prefer that the behavior of the model be independent of such technicalities. Thinking about events in real ferromagnets suggests that the root of the problem lies in the excessive abstractness of the model itself; the spurious antiferromagnetic state would probably not appear in a deeper and more detailed model, one that took into account thermal fluctuations in the position and energy of individual atoms, as well as quantum effects. But the option of choosing a deeper and more detailed representation is not available in the case of a computational universe, where by definition the program is already operating at the bottommost level!

Invertible cellular automata with time-reversal symmetry introduce still more complications. The standard recipe (Toffoli and Margolus, 1990) for creating an invertible automaton relies on an update rule in which a cell's state at time $t+1$ depends not only on the state of the neighborhood at time t but also on the cell's own state at time $t-1$. Thus the cell must have some explicit means of storing and recalling its past state, not just from one clock phase to the next but across a full cycle of the automaton. The strong version of the no-peeking rule would forbid such dependence on the past.

Some of the most interesting invertible cellular automata add layers of spatial as well as temporal structure. For example, the Margolus neighborhood (Margolus, 1984) (the basis of such ingenious models as the billiar-ball computer) relies on two superimposed lattices, which are active on alternate clock cycles. And the SALT automaton introduced by Fredkin (Fredkin, 2002) requires six clock phases, in each of which the automaton surveys a different neighborhood.

At this point we have come a long way from the minimalist vision of a cellular automaton with memoryless elements that locally and autonomously compute their own next states. To get interesting and well-defined behavior from the system, we have been compelled to equip each cell with a secret memory of its own history, and we have put the entire array under the control of a global synchronizing clock. And in some cases both time and space have been endowed with nontrivial discrete structures. Do these intricacies make cellular automata less attractive as a basis for building a computational universe? This depends on how highly you value simplicity, or perhaps on how you measure it.

The global clock is surely an uninvited guest in the world of cellular automata. Having taken pains to create a purely local computational physics, we then introduce a signal that must be broadcast to every point in the universe in every moment of time. What could be *less* local than that? But the damage is less severe than it might seem. Although distributing a global clock signal could well be a serious engineering challenge for the builder of the cellular automaton, the violation of locality does not carry through to the computed universe. Events at distant sites may become correlated as a result of the clock (if only in the trivial sense that they happen at the same time), but the clock signal cannot carry information from one cell to another.

The introduction of memory in each cell is a more disturbing change. The "primitive," memoryless cellular automaton is a device that one can look upon as a direct mechanization of Laplace's idea of simple determinism (Laplace, 1820). At any instant, each cell in the array has a definite state, and the state embodies everything there is to know about the cell at that moment. If you comprehend the laws of physics (i.e., the update rule) and you can identify the states of all the cells at any one moment, then the entire future of the universe is open to you. (If the update rule is invertible, then the past is also available.) Endowing the cells with an additional, secret variable—the memory of a past state or the premonition of a future one—compromises this deterministic purity.[8] No longer can you predict the evolution of the universe from its visible state; you also need access to secret compartments—you need "root privileges."[9]

A plausible riposte to this complaint is that knowing two consecutive states of a cell is much like knowing the position and velocity of a particle. We routinely define the state of a particle to include both x and $\dot{x}$—both position and the first derivative of position. The discretized analogue in the world of cellular automata would be knowing the state s and also Δs, some measure of the change in the state. Since in fact s is a nominal variable for which Δs has no clear meaning, by default we get to know $s(t)$ and $s(t-1)$. This is an appealing argument, but it does not quite dispel the mystery of a point in space that somehow remembers its past state. The position and velocity of a particle are properties we know how to measure (even if quantum mechanics puts constraints on simultaneous measurements). In contrast, there is nothing we can do to a point or a region in space that will persuade it to divulge its history. Points in space simply don't seem to retain that information. And where, physically, would they keep it? Perhaps it is also worth mentioning

[8] Of course one can redefine the state of the cell so as to include the new variable, making an n-state automaton into an $n \times n$ one, but the change accomplishes nothing unless all the states are outwardly distinguishable.

[9] Matherat and Jaekel, in Ref. (Matherat and Jaekel, 2001), discuss the interesting relations between causality, determinism and memory, all in the context of synchronous and asynchronous circuits.

that if such secret history bits do exist in our world, then by gaining access to them we could toggle the direction of time.

5. CLOSING THOUGHTS

The fact that we run into so many obstructions and obscurities when trying to build the simplest possible computational universe could be considered either disappointing or intriguing. It might be a clue that the whole undertaking is misguided. Or it might signal the existence of some principle or constraint from which we can hope to learn something interesting; there may be a good reason that the most primitive models don't work as we might want them to.

Nature offers no guarantee that our own universe will be the simplest one imaginable. Consider the famous quip of I. I. Rabi when the muon was recognized to be a heavy electron: "Who ordered *that*?" The new particle seemed to be a gratuitous embellishment, unnecessarily complicating the world for no apparent reason other than to vex physicists. But its existence is a fact not to be argued with. Perhaps the need for memory and multiphase clocks in a cellular automaton universe, or for $\mathcal{O}(n^2)$ algorithms in particle dynamics, should be accepted in the same spirit.

The one unquestionable benefit of thinking about physics from a computational point ofview is that it encourages total explicitness. But this benefit may be lost if algorithms are described only in abstract terms, as if in a high-level programming language. Most of the issues discussed earlier become apparent only at the register-transfer level or in a timing diagram. You cannot write a program for a computational universe—and fully specify its behavior—without reaching this level of detail.[10]

I want to conclude by mentioning one more example of this phenomenon. In Fredkin's SALT automaton (Fredkin, 2002), adjacent cells interact exclusively through one pleasingly simple protocol: They swap their states. This mechanism is not only maximally local but also automatically enforces a conservation law, since the exchanged state information is never altered. As an abstraction, "swap" seems like the ideal primitive operation for a computational universe. Looking at it more closely, however, there are nagging questions about how best to implement the swap operation. To exchange the values of cells A and B, it is not enough to write a pair of assignment statements, $A := B$ and $B := A$. If those statements are executed serially, both cells wind up with the same value (either the original value of A or that of B, depending on the sequence). If the statements are executed in parallel, the outcome is indeterminate.

[10] One might argue, on the contrary, that architectural details of a cosmic computer are totally irrelevant. As long as the underlying computer is Turing universal, it can emulate any other computer. Fair enough: But in that case we must be totally explicit about the details of the emulation program.

The usual remedy for this problem requires some form of auxiliary storage. With a temporary buffer T, the statements can be rewritten as $T := A$; $A := B$; $B := T$. This strategy certainly works, but as with other forms of memory, it raises the question of where the buffer T exists. If it is inside the cells, then again the state of the cells includes secret variables. If the buffer is external to the cells—somewhere in the interstices between them—then the universe is more than a cellular automaton; it has extra parts that need to be included in the description. Neither choice is a welcome addition to the model.

There is an alternative, usually known as the XOR trick. The values of two variables can be swapped, without use of auxiliary storage, by a sequence of three properly arranged bitwise exclusive-OR operations: $A := A$ XOR B; $B := A$ XOR B; $A := A$ XOR B. As far as I know this idea was first published in the MIT compendium called HAKMEM (Beeler, Gosper, and Schroeppel, 1972), where it appears as Item 161, attributed to R. William Gosper. Pehaps this triple-XOR shuffle seems just too cute and clever to be the basis of cosmic evolution at the deepest level. On the other hand, maybe no one will be surprised to learn that the most fundamental laws of physics were dreamed up at meetings of the Tech Model Railroad Club.

ACKNOWLEDGMENTS

I thank Edward Fredkin and the other organizers and sponsors of the Digital Perspective meeting for inviting me to participate. I am also indebted to Tommaso Toffoli, Norman Margolus, Gérard Vichniac, Charles Bennett, and the late Rolf Landauer for advice, instruction, and even debugging. A few passages in this paper appeared in a somewhat different form in the earlier publications (Hayes, 1984, 1999).

REFERENCES

Aluru, S. (1996). Greengard's N-body algorithm is not order N. *SIAM Journal on Scientific Computing* **17**(3), 773–776.

Barnes, J., and Hut, P. (1986). A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature* **324**, 446–449.

Beeler, M., Gosper, R. W., and Schroeppel, R. (1972). HAKMEM. MIT AI Memo 239. See Item 161.

Blelloch, G., and Narlikar, G. (1997). A practical comparison of N-body algorithms. In *Specification of Parallel Algorithms: DIMACS Workshop,* May 9–11, 1994, G. E. Blelloch, K. M. Chandy, and S. Jagannathan, eds., American Mathematical Society, Providence, RI, pp. 81–96.

Ceperley, D. M. (1999). Microscopic simulations in physics. *Reviews of Modern Physics* **71**, S438–S443.

Dodgson, C. (Lewis Carroll) (1960). *Alice through the Looking Glass.* In *The Annotated Alice* (with intorduction and notes by Martin Gardner), Bramhall House, New York. (Original work published 1871)

Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Reason,* Harper & Row, New York.

eXistenZ, (1999). Directed by David Cronenberg. Distributed by Dimension Films.

Fredkin, E. (1990). Digital mechanics: An informational process based on reversible universal cellular automata. *Physica D* **45**, 254–270.

Fredkin, E. (1992). A New Cosmogony. www.digitalphilosophy.org/new_cosmogony.htm

Fredkin, E. (2002). *Digital Philosophy.* www.digitalphilosophy.org/digital_philosophy/toc.htm

Fredkin, E., and Toffoli, T. (1982). Conservative logic. *International Journal of Theoretical Physics* **21**, 219–253.

Gardner, M. (1984). *Wheels, Life, and Other Mathematical Amusements,* W. H. Freeman, New York.

Graps, A. (2000). Amara's Recap of Particle Simulation Methods. www.amara.com/papers/nbody.html

Greengard, L. (1990). The numerical solution of the N-body problem. *Computers in Physics* **4**, 142–152.

Greengard, L., and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics* **73**, 325–348.

Hayes, B. (1984). Computer recreations: The cellular automaton offers a model of the world and a world unto itself. *Scientific American* **250**(3), 12–21.

Hayes, B. (1999). Computing science: Computational creationism. *American Scientist* **87**, 392–396.

Ilachinski, A. (2001). *Cellular Automata: A Discrete Universe,* World Scientific Publishing, Singapore.

King, H. C. (1978). *Geared to the Stars: The Evolution of Planetariums, Orreries, and Astronomical Clocks* (In collaboration with John R. Millburn). University of Toronto Press, Toronto.

Laplace, P. S. (1820). *Theorie Analytique des Probabilités*. Courcier, Paris.

Lind, D., and Marcus, B. (1995). *An Introduction to Symbolic Dynamics and Coding,* Cambridge University Press, New York.

Lloyd, H. A. (1958). *Some Outstanding Clocks over Seven Hundred Years 1250–1950,* Leonard Hill [Books] Limited, London.

Margolus, N. (1984). Physics-like models of computation. *Physica D* **10**, 81–95.

Matherat, P., and Jaekel, M.-T. (2001). Concurrent Computing Machines and Physical Space-Time. arXiv:cs.DC/0112020

The Matrix, (1999). directed by Andy Wachowski and Larry Wachowski. Distributed by Warmer Bros.

Maurice, K., and Mayr, O. (eds.) (1980). *The Clockwork Universe: German Clocks and Automata,* 1550–1650, Neale Watson Academic Publications, New York.

Mayr, O. (1986). *Authority, Liberty, and Automatic Machinery in Early Modern Europe*, The Johns Hopkins University Press, Baltimore.

Minsky, M. (1982). Cellular vacuum. *International Journal of Theoretical Physics* **21**, 537–551.

Moravec, H. P. (1988). *Mind Children: The Future of Robot and Human Intelligence,* Harvard University Press, Cambridge, MA.

Noyes, H. P., Amson, J., Bastin, T., Etter, T., Kauffman, L. H., Kilmister, C. W., and McGoveran, D. O. (2001). *Bit-String Physics: A Finite and Discrete Approach to Natural Philosophy,* World Scientific, Singapore, Series on knots and everything, Vol. 27.

Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics,* Oxford University Press, New York.

Schmidhuber, J. (1997). A computer scientist's view of life, the universe and everything. In *Foundations of Computer Science: Potential, Theory, Cognition,* C. Freksa, ed., Springer-Verlag, Berlin, pp. 201–208. xxx.lanl.gov/quant-ph/9904050

The Thirteenth Floor. (1999). Directed by Josef Rusnak. Distributed by Columbo pictures.

Toffoli, T. (1982). Physics and computation. *International Journal of Theoretical Physics* **21**, 165–175.

Toffoli, T., and Margolus, N. (1987). *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, Cambridge, MA.

Toffoli, T., and Margolus, N. H. (1990). Inversible cellular automata: A review. *Physica D* **45**, 229–253.

Toffoli, T., and Margolus, N. (1991). Programmable matter: concepts and realization. *Physica D* **47**, 263–272.

Vichniac, G. Y. (1984). Simulating physics with cellular automata. *Physica D* **10**, 96–116.

von Neumann, J., and Burks, A. (1966). *Theory of Self-Reproducing Automata*, University of Illinois Press, Champagne.

Wolfram, S. (1986). *Theory and Applications of Cellular Automata*, World Scientific, Singapore.

Wolfram, S. (2002). *A New Kind of Science*, Wolfram Media, Champaign, IL.

Wood, G. (2002). *Living Dolls: A Magical History of the Quest for Mechanical Life*, Faber, London.

Zuse, K. (1982). The computing universe. *International Journal of Theoretical Physics* **21**, 589–600.