International Journal of Theoretical Physics, Vol. 21, Nos. 3/4, 1982

Physics and Computation[1]

Tommaso Toffoli

MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts 02139

Received May 6, 1981

Computing processes are ultimately abstractions of physical processes; thus, a comprehensive theory of computation must reflect in a stylized way aspects of the underlying physical world. On the other hand, physics itself may draw fresh insights and productive methodological tools from looking at the world as an ongoing computation. The term *information mechanics* seems appropriate for this unified approach to physics and computation.

1. INTRODUCTION

Computation—whether by man or by machine—is a physical activity. If we want to compute more, faster, better, more efficiently, and more intelligently, we will have to learn more about nature. In a sense, nature has been continually computing the "next state" of the universe for billions of years; all we have to do—and, actually, all we *can* do—is "hitch a ride" on this huge ongoing computation, and try to discover which parts of it happen to go near to where we want.

This is not merely a problem of applied physics or technology. To develop a more fundamental understanding of information mechanics we shall be asking new kinds of questions about nature—possibly extending into new directions the very subject matter of physics.

Compare the act of performing a computation with that of performing a physical experiment.

In a computation (Figure 1) we have a certain selected portion f of the physical universe, the *computer*. We start f in specified initial conditions x

[1]This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract No. N00014-75-C-0661.

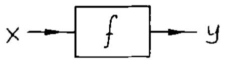

Fig. 1. Structure of a computation. f is a portion of the physical world. x and y are, respectively, initial and final conditions.

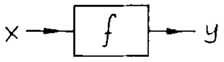

Fig. 2. Structure of a physical experiment. f is a portion of the physical world. x and y are, respectively, initial and final conditions.

(*program* and *data*); we let the machinery f run undisturbed for a certain time; and then we look at its final conditions y, which we interpret as the result of the computation.

Now, let's look at a physical experiment. Again, we have a certain selected portion f of the physical universe, the *experimental set-up*. We provide initial conditions x, we let f run for a certain time, and then we look at its final conditions, y.

Think of intelligent beings viewing us from a faraway star. How could they tell whether we are performing a computation or a physical experiment? They cannot tell by *what* we do—there is no objective difference. The difference is in our intentions, in our knowledge, in our expectations.

In a computation, we have decided that we know f, and that that indeed is the transformation which we want to apply to x to get the result y. We know f so well that we are confident we could predict its operation by pencil-and-paper figuring. But then, every time we have an x and we want to know the result y, instead of using pencil and paper we know we can let f do the work.

In a physical experiment, we have put together f, but we are not sure of how it works—that's what we want to find out! By using f several times, and comparing each time the output y with the input x, we hope to gain some ability to predict f's behavior. And when we've gained enough confidence in our predictions, we are no longer performing a physical experiment—we've got a computer running! It may not be the particular computer we want for a certain application, but a computer it certainly is.

Thus, in a computation the unknown—the thing we are curious about—is y; in an experiment, it's f. What actually happens is the same; it's only what we are going to do with it that is different.

It's no wonder then that many of the issues that we face when we probe the workings of nature at a very fine level of detail will reappear in some disguise when we try to use the workings of nature at the same fine level of detail, say, for computation. As theoretical physics had to evolve a new language to deal with these issues, so too will theoretical computer science.

2. WHAT COMPUTATIONS? WHY COMPUTATION?

What circumstances are leading us to make these new demands on the theory of computation? While the theory evolved as an abstract branch of mathematics, now we are realizing that much of the computation that we want to do is characterized by an important constraint: it must be carried out either in partnership with or as a challenge to nature. Conventional theories of computation model nature well enough to tell us *what* can be computed, but not enough to tell us *how* best to compute.

And why such an urge to compute, why such a challenge? One could beg the question by answering that these are built-in *human* traits, akin to curiosity and will. I will argue that this curiosity and this will are *humane* as well.

Substantially, there are four contexts in which we use computation.

1. *Abstract problems.* We may want to know the answer to an abstract question: What is the 100th digit of π? Or, find an English word with seven *i*'s in it.

2. *Control.* We may want to have a system capable of integrating in an appropriate way signals from input devices with the action of output devices ("When you see a stop signal, step on the brake!"). We are so use to being served by many marvelous systems of this kind that often we become aware of them only through their failures: How come it takes two weeks for a postcard to go from New York to Boston? Or, the kitchen is dirty. Has the maid been taking a nap?

Fact is, we like to have servants—strong, intelligent, obedient, discreet. There are so many things we want done but don't want to do ourselves. Greece and Rome used slaves, to work in the mines or, say, to take dictation. But we know it's not pleasant for a *person* to work in the mines, and it's boring for a *person* to take dictation. A lot of computation goes into process control; if we don't want to do it ourselves, we'll have to train some bits of nature to do it for us.

3. *Simulation.* We may want to know the consequences of a given situation: What will the weather be tomorrow? Or, how is the inflation rate going to be affected by a tax cut of 10%? In this context, we would like to have our own domesticated version of the world in a box, to play with and do things to, and yet be able to say, Oops, I'm sorry, I'll take it back—I didn't really mean it!

4. *Optimization.* Finally, we can insert context 3 into a goal-oriented servo-loop, and ask questions like: Which is the best place to spend my

vacation this summer? Here, instead of a direct problem (What will happen in this situation?) we have what is called an "inverse" problem (In what situations will such a thing happen?), which is usually much harder to solve since it may involve an exhaustive search. To find the best solution, I have to construct not one but innumerable toy worlds—one with me lying on the beach in Spain, one with me climbing the Himalayas and maybe falling off a cliff, one with me winning in Monte Carlo, or going bankrupt in Monte Carlo, etc.—and select the one that maximizes my expectations.

In context 3, we "only" try to beat nature at its own game. We want to make a weather that runs a little ahead of the real weather, an economy that runs a little ahead of the real economy. Yet, what nature achieves with a lavish outlay of resources—sun, water, lightning, and thunder on a global scale—the same things we want to safely model in our own room (we already do a lot of this every day in our own mind), with much more limited resources.

If this sounds like a tough proposition, look at context 4. There, we still want to beat nature at its own game, but many times over. And we'd better learn to do that well. For, nature has devised a wonderful scheme to get a better thing out of a good thing; it's called Darwinian evolution. You start with a "good thing," you make many more like it, only a bit different. If one of these looks "better" you keep it; the others, you throw away. It looks like a splendid scheme, and it is, but it has a catch: all is well as long as the things that get "selected against" are bacteria or velocipede patents; but not when it's things like you and me. *I* don't like to be in the test tube that gets thrown away, and I assume that *you* don't like it either. Beyond a certain level, the evolutionary scheme's simplicity is bought at a huge price—that is, untold suffering.

We want to know whether the future that we are making is good for us to live in—*before* we are going to live in it. If we don't like it, we want to make a different one. Until somebody comes up with a more brilliant scheme, we will have to play the "mutation and selection" game with cars, cities, TV shows, and social structures; shouldn't we attempt to rehearse the "selection" act with models, *before* putting real people on the scene?

In the long run, that's what computation (or thinking, if you want) is about. We are going to need a lot of it.

3. WHY PHYSICS?

We all know that computation is an "abstract" process. Whether embodied in flesh or metal, what counts is the pattern—the "form," not the "substance." We know that "128 + 128 = 256" is essentially the same process whether done in my head, or with pencil and paper, or on a hand calculator. We've studied that *with proper encoding and decoding* all effective processes can be reduced to the mill of Boolean algebra, to a network of "yes/no" switches. The instruction set of an IBM/360 can be realized indifferently, and isomorphically, by me with pencil and paper, by a hydraulic, pneumatic, electric, electronic, or neuronic computer. The physics of it doesn't make any difference to the logic of a computation; it just affects certain material aspects, such as speed, volume, and energy dissipation. Then, let the *technologist* figure out the fastest, smallest, and most efficient physical realization. What does theoretical computer science care about physics?

Well, for one, automata theory is much more closely tied to physics than is usually recognized. The very axioms of computability and computation-complexity theory are a stylization of certain physical constraints, as Turing and von Neumann (the originators of the two "standard" paradigms of computation, i.e., the Turing machine and the cellular automaton) were well aware. And this is good precedent for a young mathematical discipline. Other successful performers, such as geometry and differential equations, owe much for their productive, independent career to a long apprenticeship with nature.

Let's forget about history, and go back for a moment to context 2 (*Control*). It is true that any control task can be conceptually reduced to two stages (Figure 3a), that is, *interfacing*, and *abstract computation* performed by a single central processing unit. Thus, with proper interfacing any control problem can be reduced to a mere programming problem, where all we have to deal with is uniform, abstract symbols—rather than pulleys and muscles, pipes and gages, or electrons and crystals.

But, having written the program, built a computer to run it, and interfaced the computer to sensors and actuators, have we solved the original problem? When we put a computer inside a mechanism, the geometry and the physics of the computer cannot be neatly separated from those of the mechanism itself. The computer's weight and bulk, its power needs, the length of its wires become significant parameters of the mechanism. The idealization of a computer as "pure logic" fails. I'll give an example.

Biologists have discovered a curious fact. For a mollusk, the octopus (Figure 3b) is a smart one. It has eyes comparable to ours, and is fairly good at visual discrimination tasks—in other words, at remembering things it has seen. The octopus is also very deft at manipulation tasks. Of course, you'll say, it must be. What, with eight arms, all those suckers, and a virtually infinite number of joints (or *degrees of freedom*, to use a technical word), how can it help not be the best one-man-band around? Yet, for all of its touching talents, the octopus is poor at remembering things it has touched. How come?

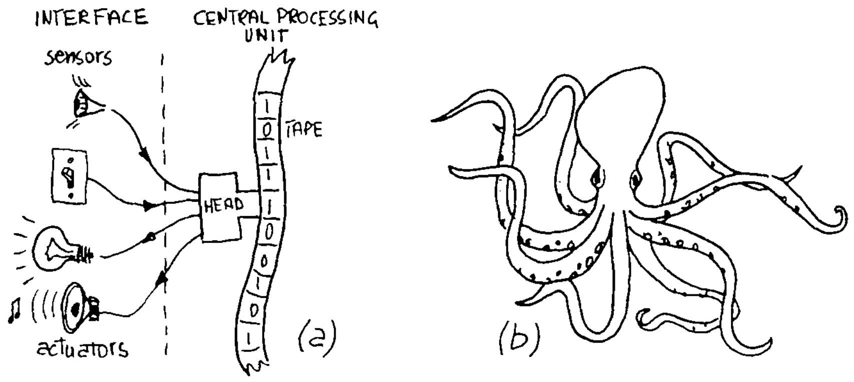

Fig. 3. (a) Through a suitable interface, a Turing machine can carry out any control task. (b) A typical example of a control task—the octopus.

Let's try to design an octopus according to the scheme of Figure 3a. We want the octopus to remember visual patterns, right? In the retina—the visual interface—we convert light stimuli to standard electrical signals, and then we run wires from the retina to the central processing unit, that is, the brain, as in Figure 4a (we may assume that there is plenty of memory space in the brain—the details don't matter). Fortunately, the eyes are right next to the brain, and so the wiring is easy. Now, we want the octopus to remember touched shapes, that is, how it feels in its arms and suckers. Well, we put a sensor on each little bit of arm and sucker, and from each sensor we run a wire to the brain (Figure 4b). But this is not the octopus we wanted. It's a bursting ball of wires with little bits of octopus just barely showing here and there!

In the real octopus, there just isn't enough room for all those wires. And what size of brain would the octopus need for thorough, independent control of the dynamics of a million joints?—We barely manage, with our large brain, to control a few dozen. So, a compromise has to be reached; in the octopus most control is delegated to tiny peripheral processors strung along the eight arms, and the brain is never told what exactly the arms are doing. No wonder it can't remember the shape of things touched!

You see what kind of reasons force abstract computation to come to terms with physics in the context of process control.

Come, now!—you'll say—the octopus has to swim and hide in crevices; it's clear it can't carry around too many extra pounds of brain and wires. But for *Simulation* and *Optimization* (contexts 3 and 4 above) we can make a huge computer and put it in the basement. It doesn't have to move—it just has to sit and think.

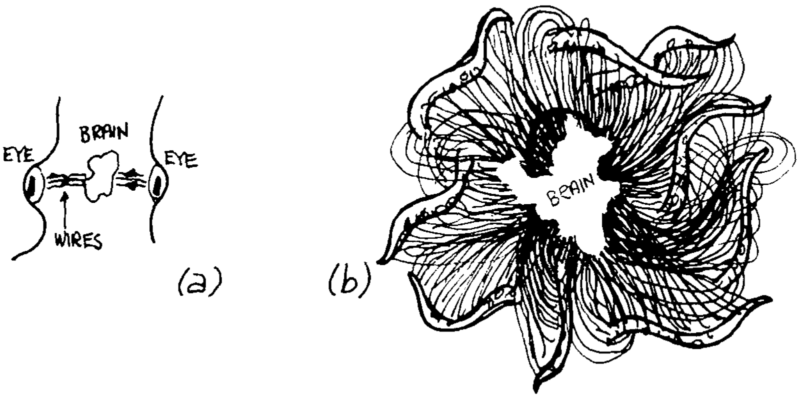

Fig. 4. (a) Wiring the retina (sensors) to the brain (central processing unit). (b) Wiring the suckers, etc., to the brain.

Here, we have a more fundamental problem. Computation dissipates energy—that is, turns high-grade energy into heat. The rate of heat removal is essentially proportional to the free *surface* of the computer, while in today's computers the rate of heat generation is proportional to the number of switches, or *gates*, and thus to the computer's *volume*. If we make a computer bigger and bigger by increasing its size in all directions, its volume will increase much faster than its surface, heat removal will become inadequate, and the computer will fry. Well, then we will make it grow only in two dimensions, keeping it flat and thin. But then certain parts of the computer will be so far away from others that even at the speed of light signals will take an inordinate amount of time to make the trip. We lose speed.

This is the single major limitation that high-performance computing faces in the near future, and it's an enormous limitation. Can it be solved by mere technologic ingenuity? No! As Landauer (1961) has made clear, the dissipation of a well-defined amount of energy comes as an inescapable physical consequence of certain very acts of symbolic manipulation. For example, in any conceivable computer, clearing one bit of memory must be accompanied by the dissipation of at least kT of energy (k is Boltzmann's constant, T the absolute temperature).

Reversibility and Dissipation. Suppose we want to add two integers. This is an *irreversible* operation. In fact, $5+3=8$; but also $4+4=8$, $7+1=8$, etc. Thus, there are several sets of initial conditions ("5 and 3," "4 and 4," etc.) that lead to the same final conditions ("8," in this case). On the other hand, the laws of microscopical dynamics are presumed to be strictly *reversible*: starting from distinct initial states one always arrives at distinct final states (by the way—I'm convinced this is all there is to the second law of thermodynamics). Thus, we have a mismatch between what we'd like to do—say, add two numbers—and what nature is willing to do for us. We cannot *just* add two numbers; something else must happen at the same time—or, better, *will* happen—whether we like it or not—to preserve reversibility.

An irreversible operation "erases" information. If we have "5 and 3," with this information we can construct the result, "8." But if we have "8," that's not enough to reconstruct the original data. It could have been "5 and 3," "4 and 4," etc. In adding the two numbers we have thrown away some information. In ordinary computers, the information that at any stage of the computation is erased from the *mechanical modes* (i.e., those degrees of freedom in which symbols are encoded) is actually dumped into the *thermal modes*—the heat sink. But every time that we open the door from the mechanical to the thermal modes to do some "garbage dumping," we have to put up with the thermal modes throwing garbage at us, in the form of noise (in microphysics, there are no one-way doors!). The thermal modes' rotten eggs are of size kT; if we want to maintain our aplomb under this barrage, the wares (and thus the refuse) of our trade must be bigger ($E > kT$). Thence Landauer's result.

To avoid energy dissipation, if we cannot change physics perhaps we can change our ways of manipulating symbols. Can we do any useful computation without destroying information—without producing thermal garbage? In other words, can we make a reversible computer, and can a reversible computer do everything that a conventional one can? The results of Bennett (1973), Fredkin (see Fredkin and Toffoli, 1982), and Toffoli (1977) show that this is in principle possible. Other obstacles may crop up on the way, but at least one conceptual obstacle to nondissipative computation has been identified, and a bypass route indicated.

4. MYSTERIES OR MONSTERS?

The medieval mapmaker, when he drew the sign HIC SUNT LEONES ("And here are wild beasts") really meant "I'm sorry, I haven't been there; how could I possibly know?"

Computer scientists tend to have a similar reaction to the many mysteries that physics holds for them. With some thinking, almost anyone can put together a few physical constants and a little dimensional analysis, and come up with good-sounding names for lurking monsters: $E = kT$, the "thermodynamical barrier to computation," or $E = h/t$, the "quantum barrier," or $E = mc^2$, the "relativistic barrier." I'm not saying that this is not a good starting point—but *somebody* must actually go out and look. Landauer, for one, spent many years stalking the $E = kT$ monster, and came back with pictures full of detail. Now we know it actually exists, we know where it lives and what it looks like; and Bennett, Fredkin, and Toffoli are softly walking around it—they think they've found a way not to wake it up.

Today, Landauer (1982) is challenging us to come back with pictures of the "$E = h/t$ thing." Only direct witnesses, please! Where does it live? Does it actually *eat* energy, etc.? We need good mapmakers willing to do their own surveying.

If we find actual monsters on our way, we may have to stop and think, and maybe we will realize that we didn't have to go through there after all. Certain obstacles may be more a matter of definition or interpretation. We all knew that the NAND gate is a universal computing primitive—anything that can be done with other primitives, the NAND gate can do. Yet, the Fredkin gate (Fredkin and Toffoli, 1982) can do things that the NAND gate can't. How come? What has changed? Who was wrong? No one—It's just that "doing" things now means something a little different, and possibly a little more interesting.

5. WHO'S AFRAID OF THE BIG BAD REAL NUMBERS?

And now, let's confess some insecurity. We computer scientists turn to physics to know what can and what cannot be done in the ways of computing. Physicists listen, smile, give us answers; they are about to leave, but then they stay for a while. They too seem glad to have somebody to talk to. They like to buttonhole us on a number of issues that have been nagging them.

How come, they say, differential equations don't seem to work so well any more? How does it happen that we take a real number, throw away most of it (when we put it in a fixed-size register), and yet get more or less the right results? Are discrete systems just an earthly reflection of perfect Platonic ideas—the continuum systems? Or perhaps discrete systems are the Platonic ones, and the continuum formalism is just a convenient way to handle the "easy ones" of them?

Tell us a story—they ask—about synchronization, incompleteness, universal constructors, algorithmic entropy. Tell us parables from informationland.

After all, what they inherited from von Neumann was something for grown-ups, i.e., a proof of the "impossibility of hidden variables" in quantum mechanics; what we computer scientists inherited from von

Neumann was the universal-computing and -constructing cellular automaton—a parlor game. Yet, within a cellular automaton there is room for both observers and observees under the same rules of the game. The system is circularity proof: it keeps working no matter how (or whether) you care to interpret it; even more, "beings" within a cellular automaton can set up their own internal cellular-automaton game, with exactly the same rules—they can maintain that they completely understand their world. On the other hand, the rules of quantum mechanics tell us something of what one part of the world looks like to the other part; they don't tell us what the whole world might look like to an "outside viewer"; they don't even help us imagine such a view.

Is there a way we can put together the two halves of von Neumann's inheritance?

We can no longer treat the physicist as white-collar mechanic: Figure this out for me, Fix this for me, Do this better for me. ("I know what I want but don't want to get my hands dirty.") Because, even as we were getting accustomed to the idea that "the world, down there, is ultimately some sort of machine," they were beginning to suspect that "the world, down there, must basically be some sort of computer!"

6. CONCLUSIONS

In the Age of Reason, mathematicians and physicists took great pride in their ability to solve problems. More recent is the fashion (undoubtedly stimulated by a number of bad encounters, such as Russel's paradox or the uncertainty principle) to prove that certain questions are unsolvable (cf. Gödel's proof and much work in computational complexity). Having verified some of our talents and digested some of our limits, we are in a better position now to tackle certain problems that are not "solvable" or "unsolvable." Rather, they are to some extent self-referential or *circular* (cf. Hofstadter (1979) and Dawkins (1976) for admirable presentations), and as such they don't ask to be solved—they have to be understood and *lived*. In my opinion, "physical limits to computation" and "computational models of physics"—the two poles of this conference—encompass one of the deepest and most vital of these circular issues. To paraphrase Feynman (1982) what else can we use to make our models of the world but pieces of the world itself?

REFERENCES

Bennett, C. (1973). "Logical reversibility of computation," *IBM Journal of Research and Development*, 6, 525–532.
Dawkins, R. (1976). *The Selfish Gene*, Oxford University Press, New York.

Feynman, R. (1982). "Simulating physics with computers," *International Journal of Theoretical Physics*, to appear.

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

Hofstadter, D. (1979). *Gödel, Escher, Bach*, Vintage Books, New York.

Landauer, R. (1961). "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, **5**, 183–191.

Landauer, R. (1982). "Uncertainty principle and minimal energy dissipation in the computer," *International Journal of Theoretical Physics*, **21**, 283. (this issue).

Toffoli, T. (1977). "Computation and construction universality of reversible cellular automata," *Journal of Computer and System Sciences*, **15**, 213–231.

Wheeler, J. (1982). "The computer and the universe," *International Journal of Theoretical Physics*, to appear.