International Journal of Applied Research on Information Technology and Computing (IJARITAC), 50-76

Indianjournals.com

Computation, Information, Meaning. Anticipation and Games

Mihai Nadin

Ashbel Smith University Professor,
Director, and – Institute for Research in Anticipatory
Systems University of Texas at Dallas, AT -10, 800
West Campbell Road Richardson TX 75080-3021
Hanse Institute for Advanced Study, Delmenhorst, Germany
E-mail: nadin@utdallas.edu

Technology is more, much more, than what people call *tools*. It embodies knowledge pertinent to the making of things (material or not). Technology is the shorthand for the *what* and *how* of our actions, whether our making involves body, mind, or both, tools, materials, energy, etc. Consequently, technology reflects our view of the world; moreover, it influences the understanding of ourselves, and even the further development of technology. This refers, of course, also to computations, regardless of whether computers are involved or not in processing them. The succession of epistemological metaphors can inform our understanding of the process of knowledge acquisition.

The ancient concept of the soul (in Chinese, Indian, or Greek belief systems) relies on experience with water, with fire, with the elements. For those involved with the primitive technologies based on these phenomena, the soul is either a pneumatic entity, or a "flame"; or it is made up of elements. The famous four *humours* (associated with Hippocrates (460–370 BC)—blood, black bile, yellow bile, and phlegm—could very well serve as a medium for the "wet" molecular computing model (eventually conceived by Adleman [1994]), so different from silicon-based computers. Closer to our time, hydraulic machines and steam engines suggested a different understanding of how the human being functions. The clockwork prompted the adoption of the machine metaphor as representative of the human being. Descartes [1637] and de la Méttrie [1748] wrote about it. For all practical purposes, their metaphor, many times refined, is fervently practiced today. The telegraph—which few of the current cell phone users will know of—suggested networks of wires. Helmholtz [1878] was probably inspired by them in conceiving the neural metaphor based upon which, much later, a "Helmholtz machine" was built [Hinton *et al.* 1995]. Shannon himself, in his MIT Master's Thesis [1940], constructed a beautiful maze of electric switches through which a "learning" mouse would run. He advanced a metaphor based on the technology *du jour*. The *MIT 150* exhibition makes the following text available:

> *Built with his wife Betty, Shannon's maze was an elegant display of telephone switching technology. When you make a telephone call, information travels the telephone system labyrinth to find the right telephone to ring, just as the mouse in this maze searches for its cheese. (cf. http://museum.mit.edu/150/20)*

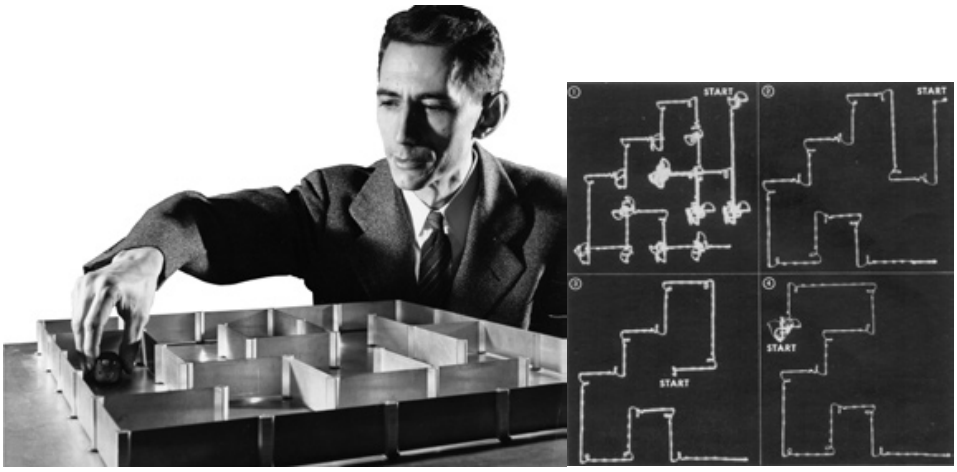


Figure 1: ***Shannon's maze. A symbolic analysis of learning based on relay and switching circuits.***

Packet switching [Baran 1964], with its variable-bit-rate data streams over a shared network, was still far away. Shannon's metaphor, the "computational brain" (originally the McCullogh-Pitts model [1943]), and Turing's [1950] elaborations on the mind as a machine belong to the same family (as different as they are in details).

Parallel to this technology-driven development that posits computation as the metaphor for all that is, mathematicians identify their own perspective with reality. Mathematical descriptions are yet another form of a machine: an abstract machine. Nevertheless, their description of the universe [~~cf.~~ Kepler 1609]) can inspire the metaphor of a celestial machine. Much later, Feynman [1960] suggested the nano-level metaphor: "There's plenty of room at the bottom." In years closer to our time, Rosen [1991] took note of the fact that the Newtonian *clockwork* world view refers to three entities: *the figures, the gears*, and *the particular rules for arranging these gears and springs*. Implicit in this observation is the concern for understanding how time, in its generality, is reduced to a "machine," and how this machine—the clockwork—becomes a comprehensive model of the world. Einstein [1916] pursued the clockwork metaphor, establishing yet another perspective: relativity theory. The following summarizes his views: In Einstein's theory, "There is no single…universal clock;" "You can still tell a clockwork like story about the evolving universe. The clock is your clock. The story is your story," [cf. Greene 2004].

Today we look at all that is and "see" computation—even the clock is a computer. But does it really make sense to ascertain that computation is universal after having discovered that previous metaphors—the hydraulic machine, clockworks, steam engines, neural networks, etc.—were only steps in understanding our reality and ourselves? Scientists, philosophers, engineers, and others warn that every *final* metaphor will prove to be as partial as the previous metaphors. As a matter of fact, in every instance of scientific and technological development, several metaphors are simultaneously at work. Particle physics, for example, inspired scientists to build models of crowd dynamics. The reason is simple: There is already a knowledge base—the mathematics of particle behavior—that can be used. However, the computation metaphor is comprehensive in the sense that it suggests that, in keeping with the example given, the particular crowd behavior is also the result of some computation.

This paper cannot make the conflict of succeeding metaphor sketched above its subject. But it can examine the extent to which the new "machine"—probably less "new" than some might think—addresses fundamental aspects of knowledge. The focus here is on *timeliness*, i.e., how much future, as a dimension

of time, is possible in the process called *computation*. The Turing-based machine, corresponding to an abstract mathematical machine, suggests that in understanding computation, we do not need to focus on implementation. Those interested in *new* forms of computation have sought inspiration in the most complex aspects of reality, that is, in the dynamics of the living. The diagram below (Figure 2) illustrates (but by no means exhausts) types of computation embodying metaphors that guide science today.

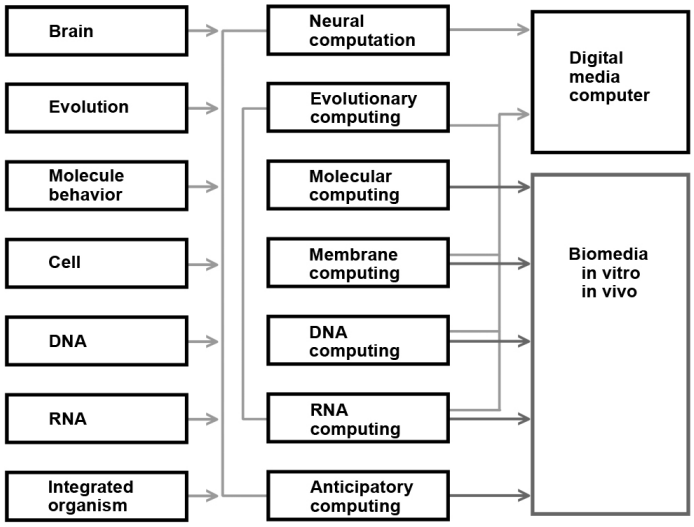


Figure 2: ***Types of computation inspired by metaphors pertinent to the living***

Many other types (e.g., nano, quantum, adaptive, proactive, autonomic computation, etc.) are rapidly populating the field. Nobody can keep up with all these new developments. Their multiplication corresponds to the rapid broadening of the scientific perspective, i.e., the multiplication of questions regarding the dynamics of reality.

Obviously, the circularity of the argument—which constructivists noted early on [Foerster 1976]—is evident: If you ascertain that the brain (for example) works like a computer, neural computation will only confirm the premise. There is nothing to falsify [Popper 1959] since the approach is epistemologically circular. The diagram is useful because in the discussion to follow, all these forms of computation (as well as those left out) are implicitly considered, even though, in the main, von Neumann's model of the sequential computer will be considered. Parallel computation and neural networks will also be present in the line of argument.

1. PROGRAMS ARE MACHINES

A computer program (henceforth, program) is a machine. There is no way around this condition of programs. But until we understand what this entails, the statement bears as much knowledge as any other truism. No programmer can benefit from it. Can we give this statement a little more practical significance?

As much as programming, in its many varieties, has advanced in the last ten years, it remains an ill-defined activity—something between engineering and art. (Dijkstra [1976] wrote convincingly about this.) A better understanding of the goals and means of programming might facilitate alternative ways to evaluate its outcome, including the debugging that burdens users beyond everything experienced in the past. If cars, airplanes, even household appliances were produced with a similar proportion of errors as software programs, the number of accidents (and damage lawsuits) would be staggering.

If a program is a machine, what are the consequences for our understanding of the *time dimension* of programs? The question has pragmatic implications: Today we are engulfed in programming more than in any other form of human activity. A growing number of people earn a living in this field; and a greater number of people make a living using programs. Behind almost all activities—datamining, security, medicine, art, games, entertainment, and the production of goods, drugs, machines, processed foods, new materials, and new forms of energy—programming is involved in a broad range and ever expanding variety of forms. We invent new materials in computational form before we actually "make" them; we explore new medicines and new procedures; we design the future (architecture, urban planning, communication, products) using programs; we redefine education, politics, art, and the military (or wars) as we express our goals through programs. Terrorism is "programmed;" so is security—the most widespread application of computation, affecting almost the entire population of the world. All the invisible computers (embedded in our world) that populate our universe of existence (from the watch we wear to thermostats, speedometers, appliance controls, cell phones, pacemakers, etc., etc.) have been programmed and keep undergoing reprogramming. They serve in meteorological, seismic, oceanographic, logistic, and transportation applications. More and more, they serve in production-related activities: monitoring the functioning of automated production lines and the reliability of critical components of such lines. A deformation in steel production (rolling) can cause a world-wide crisis, given the tight interlocking of steel availability (production) and demand (*just-in-time* translates as high loss potential if the steel is not delivered as expected; cf. Corts and Hackmann [2009]). Therefore, to address time aspects of programming is to account for the *meaning* and *efficiency* of a form of praxis that defines the human being of the present and the future [Nadin 1998].

But what does it mean *to program*? Let us take a simple program procedure: the factorial, which is frequently present somewhere in the larger scheme of things, though not of particular importance. It is part of the mathematical description of the world. The factorial of a number n is denoted by n! and is defined in mathematics as

$$n! = [(n-1) \cdot (n-2) \ldots .3.\ 2.\ 1] = n \cdot (n-1)! \qquad (a)$$

Even people who refuse to look at a mathematical formula can notice that to calculate the factorial of n, one would calculate the factorial of (n-1) and multiply the result by n, that is,

$$n! = (n-1)! \cdot n \qquad (b)$$

(Obviously, if n=1, the factorial is 1.) This means that in order to calculate the factorial, we multiply 1 by 2, the result by 3, the new result by 4 and so on until we reach n. A counter keeps track of how the numbers increase from 1 to n.

How does a computer program handle this? We can, as done above, define the factorial computationally:

Program lines	What the program lines mean
(define (factorial) n)	
(if (=n1),	which means if n=1,
1 ⟶	the value is 1
(*n (factorial (-n1)))))	multiply (*) n by the factorial of (n-1).

Figure 3: ***Program lines for computing the factorial of n***

Programmers have internalized all this and no longer pay attention to the details.

"Where is the machine here?" some will ask (and not only those with no broad knowledge of computers). As we know from literature (from systems theory, in particular), or from using machines, a machine (defined as a system) takes something, called *Input*, does something with it, and produces the result as *Output*:

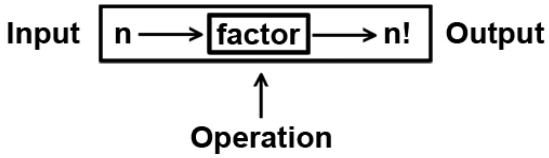


Figure 4: ***A machine called factorial: a systems perspective***

This applies regardless whether the machine crushes stones, makes salami from raw meat, or keeps a certain rhythm controlled by weights turning a wheel (the mechanical clock). It applies as well to the "machines" we call brain cell, organism, etc., if we use the knowledge metaphor that reduces the brain or the cell to a computation.

Let us consider a rendering (Figure 5) of the machine called the *factorial program*. (In this case, the input is the number 6, whose factorial will be calculated.)

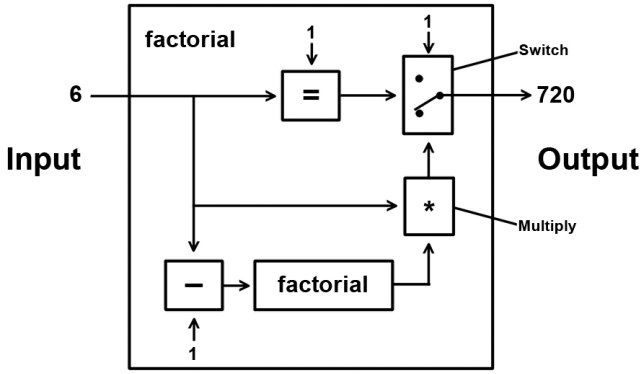


Figure 5: ***The factorial program (see above) can be viewed as an abstract machine.***

We notice here parts that decrement (6, 5, 4,...), multiply (*), and test for equality. We also notice a 2-position switch and a factorial machine (it takes a number and calculates its factorial). Obviously, our machine contains another machine, the one called *factorial*. This qualifies it as an infinite machine, in the language of machine theory.

2. TESTING THE MACHINE

Machines within machines within other machines, etc.—this is not an easy idea to grasp. For people used to the "old" machine model, something can go wrong with various parts (gears, springs, transmissions, etc.). In order to understand what is wrong, we need a testing procedure. Each clockmaker had testing procedures. Manufacturers supply each of their customers with a sequence of operations to ensure the proper functioning of the various "machines" making up a car. The conceptual machine, i.e., the program, consists of parts that are subject to tests of a different nature. If we want to have our program evaluated,

we need to submit it to an evaluator, or interpreter, that emulates the desired functioning. In simple language, this means to test it in order to find out what to expect from the program.

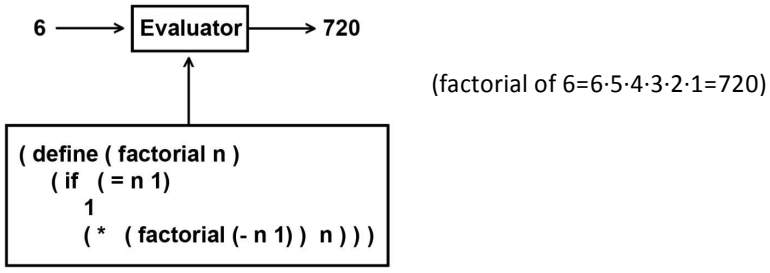


Figure 6: ***The evaluator emulating a factorial machine and producing the value for 6! (The two examples are presented here with the kind permission of Abelson Sussman, and Sussman, who wrote a fundamental book on computer programs [1996].)***

The evaluator takes as input a description of a machine (the program) and emulates (i.e., imitates) its functioning. Accordingly, the evaluator appears as a universal machine; that is, it "knows" how all programs work. Imagine such an evaluator as an entity that can look at the plan of your future house and return something like a validation stamp, or draw attention to the fact that the bathroom on the second floor has no connection to the water pipe.

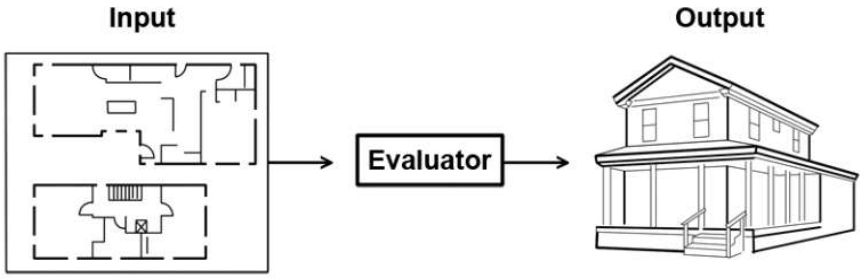


Figure 7: ***Evaluation for architecture (simplified view)***

The same evaluator can also check out your Webpage design, your new recipe for chicken soup, or the plans for a new car, and return a meaningful interpretation that will guide your actions. No human being and no computer can "contain" the knowledge, that is, store all this understanding of how the world functions. The broad scope of such diverse projects lies well beyond what individuals, or even computers, can master. But this is what it takes to validate the "program" we call architectural plan, Web design, chicken soup recipe, or automobile engineering (to name a few examples).

To evaluate an open-ended gamut of programs is impossible. Program evaluation is possible when we have a constrained domain of knowledge. Computations corresponding to the limited domain defined by the program are affected by the nature of knowledge representation in the form of programs and by what we expect from a program. To put it briefly: Gödel's incompleteness theorem [1931] guarantees that for a limited domain the description can be complete and consistent.

If we denote knowledge of the world as KW, and domain-specific knowledge as DSK, then KW contains DSK. Have you noticed any reference to *time* up to this point? No. The reason is simple: Machines are timeless; even the machine we call *evaluator* (or *interpreter*) is timeless. Unless something breaks down

in a machine, it will endlessly and uniformly perform the function for which it was conceived. The best machine repeats itself *ad infinitum* (or at least until its physical breakdown) without taking note of time. If it has a counter, the number on the counter shows "How far from infinity" it is, but not really what time it takes to produce some meaningful output. Its reason for existing is this orderly behavior, its predictability. Its underlying principle is determinism: the cause-and-effect sequence. There is, however, an implicit time factor here: Cause precedes effect. But in fact, this time factor is also reducible to a machine, more precisely, to one that measures intervals: the clock.

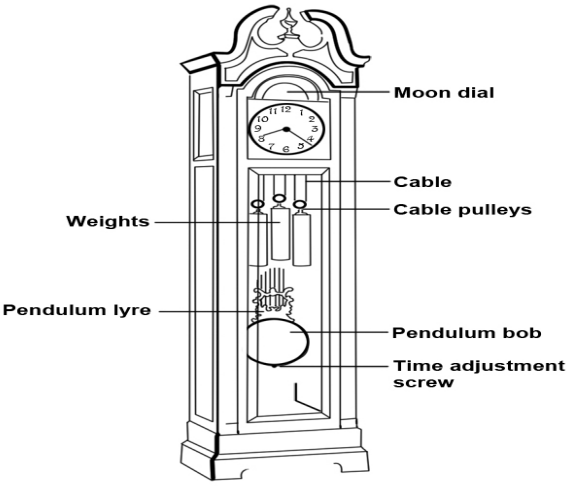


Figure 8: ***Time is not only interval. Gravity keeps "counting" seconds for us.***

That time is more than interval—just as space is more than distance—is an idea we will eventually have to entertain as we progress in our discussion of the timeliness (and *future-ness*) of programs.

Once we acknowledge determinism as the underlying principle of the human-made entities we call machines—in the form of artifacts or as mental machines—we also acknowledge that the "patron saint" of this perspective of the world is René Descartes (1596–1650). Western civilization adopted his views, albeit some ("Is the human being reducible to a machine?") slightly modified. Consequently, Western civilization adopted the rationality of his fundamental contribution—reductionism—and gave up any claim to a holistic understanding of the world. According to Descartes [1637] and others, all there is, in its amazing complexity, can be managed by breaking the whole into its parts, and then describing each and every component from the deterministic viewpoint of the cause-and-effect sequence. Within this encompassing view of the unified world, machines are a good description of the living as embodiment of a functionality achieved from lower-complexity elements.

The Cartesian conceptual revolution extends well into the computer age, although as we advance in our understanding of the difference between the living and the physical, we are starting to question some of its tenets (as Schrödinger and Bohr already did). Moreover, with the computer, the inherited notion of the machine starts being challenged: no more gears, weights, and coils, no more an exclusively energy-controlled entity, but rather one in which information (mainly in the form of data) plays the crucial role.

Obviously, this is not the place to rewrite the chronicle of determinism or to start yet another anti-determinism crusade. Neither is it the place for an account of the many questions—the holistic perspective being one—that determinism has left unanswered. However, without understanding the fundamental perspective it establishes, and the challenges we face in questioning this perspective, moreover in establishing a new

perspective, we could not answer even the most trivial questions regarding the future of computation. Indeed, in looking at the time aspects of programming, we are looking (aware of it or not) at the future of computation. Some see this future already woven into the programs we write today; others in the new technologies of computation (computing with light, DNA computing, quantum computation, etc.). And yet others see this future in a computation understood as a living entity, or at least as a hybrid entity (involving living components) able to reach anticipatory performance [Nadin 2003]. Indeed, anticipation is always a holistic expression of all the processes leading to it.

3. FROM THE HUMAN COMPUTER TO COMPUTERS

If you are not trained in the specific representation that makes these hybrid computations operational, they make no sense. Therefore, when we talk about "computer science," we are using a catch-all description for the many different sciences it includes, in a way similar to mathematics as the collective name for fields so different as differential calculus, topology, string theory, etc. One can say that the reductionist approach, i.e., the focus on particular aspects of the whole we are interested in, results in

1. the impossibility that there be a mathematician who knows everything about mathematics, or a computer scientist who knows everything about computers; and in

2. the need for specialized knowledge, i.e., data-mining, signal processing, virtual reality, etc. professionals.

In practical terms, this means that education in computation needs to be rethought in order to reflect the variety of computations currently needed, and of new forms of computations that might be needed in the future. The diagram (Figure 2) is informative of the reductionist model. It shows how various parts (brain, molecular behavior, cell behavior, etc.) are represented in computational form. For those familiar with these terms, it is evident that neural computation, implemented as neural networks, is different from DNA computing or membrane computing. As a matter of fact, in each of the computational forms mentioned, there is a different mathematics embodied in programs, that is, a different method for automating mathematical operations—actually, a different metaphor.

Early on, some of us realized that the computer, as a machine, is by no means more interesting than an abacus. We did not doubt that an automated abacus is faster than any expert in using this relatively old arithmetic machine. We did not doubt that an automated abacus could perform many operations per time unit (i.e., that it can be fast), that it could store data (even in primitive registers) beyond our own memory performance, that it could become the functional repository of all our arithmetic needs. In other words, the abacus would "know," for us, all there is to know about arithmetic.

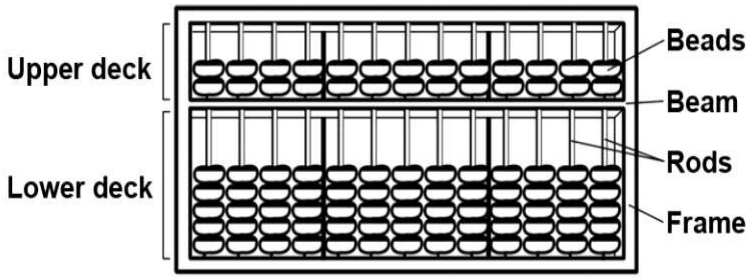


Figure 9: ***"Hardwired" knowledge in the abacus***

What really drew my attention to the machine called computer—the term *computer* was used in the 19th century to denote a profession carried out by human beings—was a totally different question: Does the abacus know arithmetic? (The human being called *computer* by reason of his profession knew arithmetic and probably more than that.) Furthermore: Does the computer know—as much as the human called computer knew—what it is processing? The "human computer" I refer to were those hundreds of people employed to calculate, for example, astronomical data [Grier, 2005]. This was a huge effort, very tedious, and everyone was looking for ways to automate it. If the hundreds of *computers* knew what they were calculating, then did this understanding of what was computed affect the outcome? (For example, the program carried out by the professionals called *computers* took the form of astronomical tables.) And again, if they knew, then how? (The *human computers* learned it, and continued learning as they were exposed to new data.) But where does the machine's knowledge come from? Before ending this personal aside, I want to add one more piece of information. My computer—the one I was supposed to study and program in the early 1960s—did not exist. It was on paper [Nadin 1955-60]. This sounds absurd today, but that was the reality in a part of the world—Romania—where the absurd was "invented." (Just recall Ionesco's theater of the absurd, and Tristan Tzara's Dada movement.) And that was my good fortune because, after all, computation is about programs, not about switches, tubes, electrons, storage, keyboard, and all that makes up the necessary, but by no means sufficient, hardware.

4. AUTOMATED MATHEMATICS: THE ANALYTIC ENGINE

Computation emerged as nothing more than a way of automating mathematics. Before digital computation, scientists tried to reach automation through means corresponding to the pragmatics of their time. John Napier (1550–1617), the Scottish inventor of logarithms, tried [1617] to simplify the task of multiplication, through his famous *rods* or *bones*. Wilhelm Schickart (1592–1635) built a machine that performed sophisticated operations; Pascal (1623–1662) devised adding machines (ca.1641–1645); Leibniz (1646–1716) introduced binary code (1703]; Jacquard (1752–1834) built the loom, able to generate complicated patterns (computer graphics before the age of computers). As we know, Charles Babbage (1791–1871) figures high in books (and stories) through his two machines—the *Difference Engine* and *Analytical Engine* [1864]—which apparently were never built, but which he worked on from 1822 until his death. So does William Stanley Jevons (1835–1882), more famous for observations about economics, who in 1869 built a machine to solve logic problems [Peirce, 1887]. Carissan (1880–1925), lieutenant in the French infantry, made a mechanical contraption for factoring integers and testing them for *primality*. And we know of Leonardo Torres y Quevedo (1852–1936), who assembled (or is famed for allegedly having done so) an electromechanical calculating device that played chess endgames.

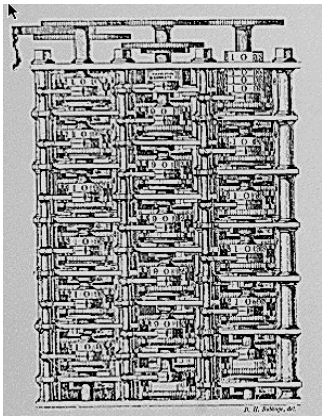


Figure 10: ***Babbage's Analytic Engine***

This short list is only indicative of an understanding of temporality that extends well beyond the current use of the word *program*. Each of the individuals—scientist or not—mentioned above programmed, but more in the sense in which the abacus is programmed, not the digital machine, which is at the heart of computers today. One can say that the abacus is "hardwired;" in other words, it is its own program. Dependence on the hardware is implicit in mechanical and electro-mechanical contraptions. Napier's rods and Pascal's adding routines are timeless. Indeed, today they would allow us to carry out operations with the same degree of precision as in the days in which they were conceived. Even in our time, the Jacquard loom serves as a model of programmed patterns, the only difference being that, in a program, we can handle more data and readjust the "digital loom" in almost no time.

Since I mentioned Babbage, two things deserve to be highlighted: He extended the meaning of the word *engine,* corresponding to the machine of the Industrial Age, to a processing unit of mathematical entities. As a cognitive instrument, the metaphor affected the future understanding of machines meant to process information. Some [Gardner 1958, Ketner 1984] attributed to Charles Sanders Peirce a computer using the electro-mechanical switches of a hotel system (pointing to rooms reserved, occupied, vacant). Peirce went far beyond Babbage, as he wrote upon the latter's death [1887]:

> But the analytical engine is, beyond question, the most stupendous work of human invention. It is so complicated that no man's mind could trace the manner of its working through drawings and descriptions, and its author had to invent a new notation to keep account of it (p. 458).

He also pointed out very precisely that "Every reasoning machine [as Peirce, 1871, called them] . . . is destitute of all originality, of all initiative. It cannot find its own problems. . . ." In our days, programs are much more complicated than when Peirce wrote his notes. The manner of their working reflects a stage in the acquisition of knowledge and the associated technology that corresponds to questions that could not even be formulated 100 years ago.

The automation of calculations for ballistics in the Mark I computer (created in 1944 by Howard Aiken and others at Harvard University), and in the artillery calculations on a general-purpose electronic machine, on the ENIAC (at the Moore School of the University of Pennsylvania) ~~are~~ is in fact the identifier for modern computation. Indeed, automated mathematics is the shorthand for the initial modern computer. Behind this not trivial observation we find the origin of almost all the questions preoccupying us today in respect to computation. This begs ~~some~~ explanation. Descartes proclaimed the reduction of everything to the cause-and-effect sequence and the reduction of the living to the machine as the embodiment of determinism. This reduction resulted in the description of time as*duration*, and of the living as *functionality* (which the machine expressed). The program of the Descartes-type of machine is given once and for all time. It does not change, as performance does not change unless the components break down. In the deterministic machine, the implicit time dimension pertains to its functioning, which is dictated by the physical characteristics of the components. Such a machine exists, like everything else in Descartes' world, in the time dimension of existence reduced to duration.

With the advent of the computer, the implicit assertion is reasonable: For the class of mathematical descriptions of the physics of ballistics, artillery calculations in particular, we can conceive of a machine that will automate the calculations. In other words, the determinism of the physics described in mathematical equations of ballistics is such that we can automate their processing. One can generalize from such equations to many other phenomena. Space exploration, as well as the trivial description of playing soccer, comes easily to mind. One can take the mathematics of the particular ballistics problem as an attempt at modeling many phenomena of practical impact. If we know how to handle such difficult descriptions, we already

know how to handle simpler cases, ranging from simulating a game of billiards, to building games driven by the same program, and to building a control device to guide a rocket. The abstraction of mathematical descriptions, to which I shall return, makes such descriptions good candidates for an infinite variety of concrete applications.

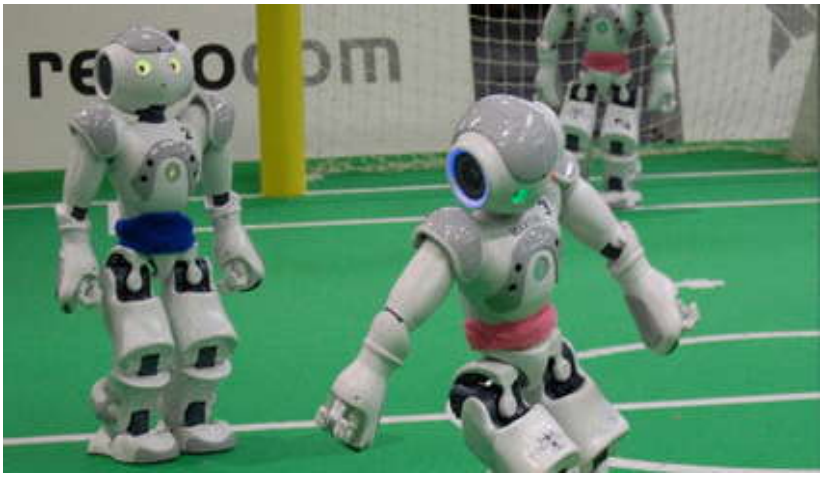


Figure 11: ***Robocup, the BreDoBrothers Team: Kondo robots performing in an AI-based behavior control environment [Röfer et al. 2006]***

This is no small accomplishment. But it is by far not yet what we understand when we use the words *computer* and *programs*. We need to be even more specific. Ballistic equations, as complex as they can get, are only a small aspect of mathematics. (Meanwhile, they have been substantially improved.) For all practical purposes, a dedicated machine (driving a cannon, for instance) is nothing more than a description of the task to which it is dedicated. The implicit assumption is that of Descartes' machine: It performs within a world that is regular, repetitive, hierarchic, centralized, and predictable. Even the variety of applications it might open is treated the same way. However, once we transcend the specialized machine and enter the domain of computation as a universal process, we transcend the boundaries of the reductionist perspective. Distributed intelligent agents can easily replace the centralized cannon control program. Such an implementation questions sequentiality, hierarchy, and centralism. A multi-agent society—an aggregate of interconnected autonomous agents—replaces sequences through computational configurations. We can, in principle, reach some levels of holistic integration. This opens up a new perspective: either to accept the model of a permanency that extends from the physical to the living and to society, or to acknowledge dynamics and take up the challenge of understanding knowledge as process.

During my research residence at the Technology Information Center (TZI) at the University of Bremen (December 2009–May 2010), I focused on endowing intelligent multi-agent societies with anticipatory characteristics. Given the fact that adaptive processes need to reflect change and to adjust their behavior, the question became how to integrate the variety of data characteristic of logistics in a manner that allows for autonomous intelligent agent activity. The prediction of change has to be coordinated with the information from sensors in real time. The decentralized structure was the option favored by the researchers involved in the project. The project's complexity is reflected in the following diagram:

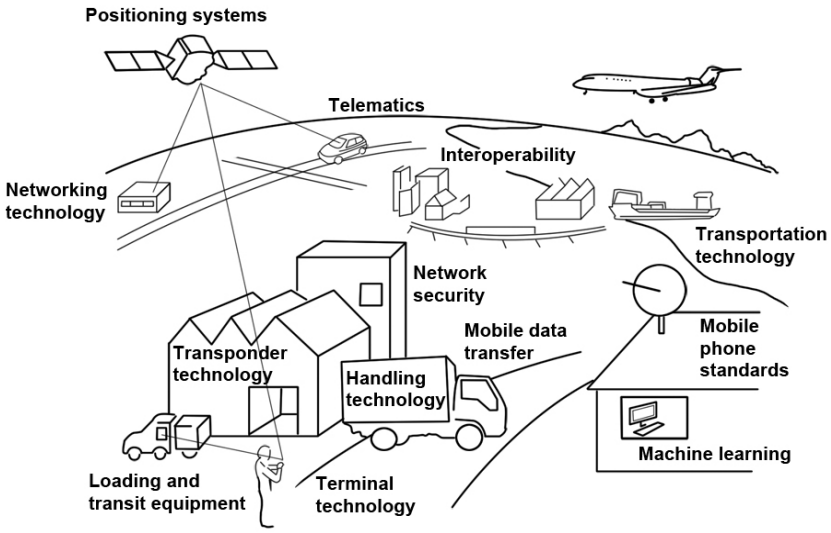


Figure 12: ***Technological advances in logistics (courtesy of the Logistics project at TZI Bremen)***

During this time, I realized the urgency of examining the relation between an information-driven model of computation and one in which meaning is simultaneously acknowledged. After all, logistic processes involve many individuals, and are based on many spontaneous actions (e.g., on implicit trust).

5. UNAVOIDABLE QUESTIONS

Among the many questions to come our way and guide our endeavor are:

a) Information and meaning—how to preserve their unity?

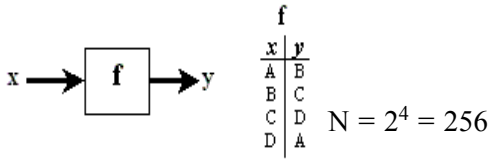
b) Is everything reducible to mathematical descriptions?

c) Is everything describable in the language of 0, 1 (precision vs. expressiveness)?

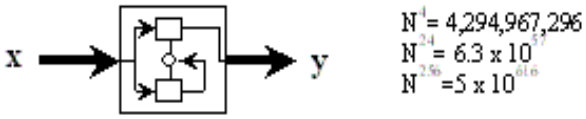
d) Is Boolean logic the expression of all there is to the logical decisions we make in life (whether it is a matter of deciding what to have for breakfast or how to understand the genetic code)?

These questions—extremely important to computation practitioners (programmers, scientists, all kinds of users, automated procedures, etc.)—will be addressed once we examine the "ingredients" of the activity.

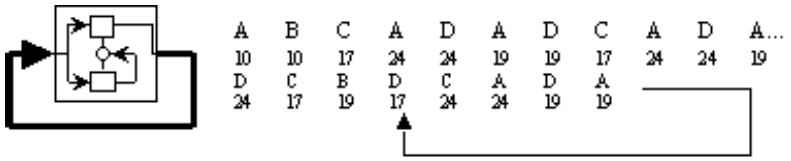
The computer understood as automated mathematics entails a Cartesian curse: If something is reducible to a mathematical description—or if our mathematical descriptions are abstract enough—it can be computed. Many scientists and engineers live by this notion. They simulate life in computational form and study it as though it were real, as real as our skin, pain, birth, death—only on a more global scale. Others draw our attention to a simple realization: Our descriptions, regardless of the medium in which we produce them, are constructs. Their condition is not unlike the condition of everything else we construct, subject to physical limitations, but also nothing more than products of our minds. Such mind products are focused on understanding the context in which our individual and collective unfolding as human beings takes place. Understanding of the living as a machine is only one among many. Von Foerster [1995] tried to construct a machine typology (Figure 13: a, b, c).



13a: ***Trivial Machines: "(i) Synthetically determined; (ii) History independent; (iii) Analytically determined; (iv) Predictable."***



13b. ***Non-Trivial Machines: "(i) Synthetically determined; (ii) History dependent; (iii) Analytically indeterminable; (iv) Unpredictable."***



13c. ***Recursively Operating Non-Trivial Machine: "Computing Eigen-Values, Eigen-Behaviours, Eigen-Operators, Eigen-Organizations, etc..."***

These very instructive descriptions of some machines are indicative of the constructivist perspective. They are explanations that human beings produce as they try to explain the world they live in. Indeed, we can construct artifacts that can operate for us on a variety of representations (e.g., numbers, words, sensory input, images, sounds). The operations correspond to fitting the mathematical descriptions to the reality described. A probabilistic machine, such as one based on nanoscale digital circuits [Riedel 2009], will have probabilities as inputs and outputs encoded through statistical distribution of the signals. Probabilities are mapped into probabilities.

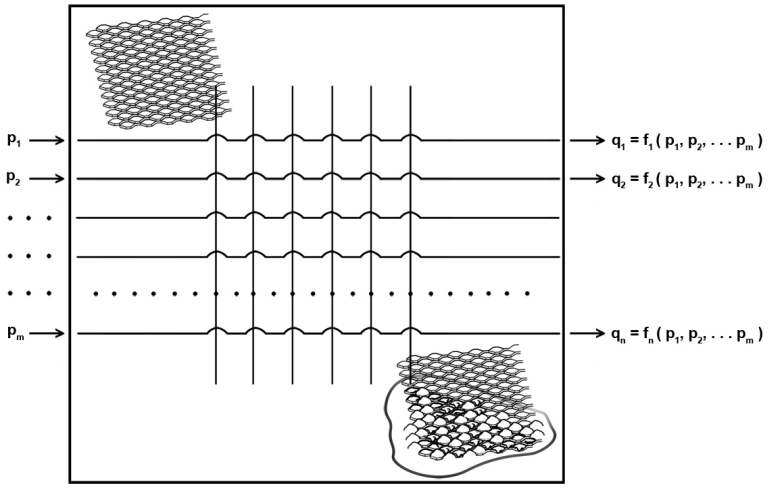


Figure 14: ***Computation for nanoscale technologies (cf. Riedel)***

Today we call the mathematical description clause a *necessary condition* for something to be expressed through computation as we know and practice it. We add to it the expectation of a logical description. To be *computational* means to be expressed through a computational function. The time factor renders this condition ultimately insufficient. Imagine that we have captured whatever is of interest to us (for our work, enjoyment, inquiries, etc.) in computational form. And further imagine that we have enough computational resources (memory, processing speed) to process our data. Time, in the sense of duration or interval, limits the endeavor, because many computational functions are *undecidable* or *intractable,* which basically means that it will take a time interval beyond what we dispose of (the lifespan of an individual or a generation) to perform the computation. Complexity has its price. Descartes was in no hurry. For him the clock's rhythm, translating the natural rhythm of the day–night sequence, was fast enough to guide his descriptions of the living as being no more than a machine that processed sensorial information. Now that our clocks have become very fast—the beat of the digital engine at the heart of our desktop machines exceeds the gigahertz level—we hope to recover some of the complexity that in "slower" times was done away with. Still, the most intriguing questions we would like to address remain outside the domain in which automated mathematics and automated logic (sometimes called *reasoning*) support many of our current theoretical and practical endeavors.

This brings up the more probing question of the language describing our inquiry. If we want to acknowledge time, we have to deal with *process*. The word describes a dynamic entity: something taking place over time. And as it takes place, it affects something else. In the mechanical age, to process meant to affect the physical and chemical appearance of substances we wanted to change, preserve, combine, or extract. Let us take note of the fact that some [Fraley 2010] declare computation, in its generality, to be *process*. On this path, we can overcome the need to acknowledge an open-ended typology of types of computation. We can also transcend the understanding of algorithmic computation. The danger is—as is usually the case when generalizations are attempted—to see "everything that is going on" (the "going on" corresponds to a simple definition of process) as a computation. If everything is a computation, we no longer have a reference—and therefore epistemologically there is nothing to gain from the statement. The practical consequences are nil: Those who program will not do better or worse by thinking that computation is a process. However, we can focus on the particular type of process, or processes (since there are so many different kinds) involved in computation as we know it so far. The emphasis will change from the noun *process* (plural *processes*) to the verb *to process*. Today the verb *to process* applied to computers has as its object an abstract entity called *information*. Indeed, computers can be understood only in association with the object of their functioning, only as processing data (the concrete form of information). Therefore, it is equally important to define the object of the process, i.e., information. To ignore that our notion of information, as we use it in computer science, stems from thermodynamics (cf. Shannon's definition [1948]) means to move blindly through a world that is constructed on the very premise of our acceptance of the laws of thermodynamics.

5.1. INFORMATION AND MEANING

A closer look at how information is defined might tell us more about the time dimensions of programs than the programs themselves. Shannon's genius (and direction) is probably comparable to that of Descartes. He considered information strictly from the engineer's perspective: Give me an input that my machine can expect, and I will make sure that the processing of this input will not alter it beyond recognition. His focus was on communication; and accordingly, he did not concern himself with anything except the physical properties of the carrier. Meaning is ignored, which means that, specifically, the semantic dimension is of no concern. What I describe here is well known; I myself have addressed this issue of the exclusive use of syntax more than once [Nadin 1982]. But there is one more thing to be added here: Syntax,-as we

know it from semiotics (to which I shall return), is timeless. It captures only the description of the carrier, not the meaning of the message, and even less the pragmatic dimension. Working for the Bell Telephone Company, Shannon was concerned with the price of sending messages through a telephone line. He noticed that a great deal of what makes up a message is repetition (what we call redundancy, i.e., that part of what we exchange in communication that carries nothing new with it). Actually, for Shannon, information was the inverse of redundancy. If prior to reading these lines your knowledge of Shannon's theory was that a) "He was the founder of information theory;" and if after reading these lines you double your knowledge to b) "Information theory is reductionist theory," then I contributed a bit (pun intended) to your knowledge.

On 10 March 2004, the American public was glued to their television sets watching the outcome of a celebrity trial (Martha Stewart, a household name in the USA, was on trial before a jury). It was a typical Shannon experiment. There were four counts on which the jury had to render a verdict of guilty or not guilty. Outside the courthouse, thousands waited to hear the outcome. TV cameras from around the world focused on the exit from the Manhattan courthouse. And as the foreperson was reading the jury's verdict, strange messengers started a bizarre show. They ran ahead waving above their heads colored scarves—red for guilty on count 1, blue for guilty on count 2, etc. The TV viewers had no access to details of the color code prepared in advance by journalists hurrying to be the first to make the verdict known. Prior uncertainty—guilty or not on count 1, etc.—was halved each time a runner with a colored scarf ran down the stairs. Ultimately, if the jurors themselves had been on the stairs and used all the sentences read inside the courtroom, the information would have been the same. The text they would have read would be informationally equivalent to the color of the flag. One *bit* is defined as the information needed to reduce the receiver's uncertainty by half, no matter how high that prior uncertainty was. It is a logarithmic measure, and the formula behind the whole thing is

$$\mathbf{H = - \sum_{i=1}^{M} P_i \log_2 P_i \ (bits\ per\ symbol)} \qquad \mathbf{(c)}$$

This says that information, defined on the premise of the reductionist machine model, is commodity, quantifiable like energy consumption or like currency flows. In this respect, every program that is based on the assumption that entropy (a concept originating in thermodynamics and describing the disorder of a system) is a good model for information dynamics remains fundamentally in the realm of physical entities and their respective deterministic laws. This is the model of the carrier (the sign) reduced to its appearance (the syntax).

In the realm of the living, which always includes the physical but is*not* reducible to it, entropy only partially qualifies the dynamics of the whole. Accordingly, for all programs pertaining to artificial machines, Shannon's information theory is an appropriate foundation. But once we enter into living computations, or into the promising hybrid computation (living and artificial in some functional connection), the notion of information itself is no longer adequate. With the living, time, in its richness—that is, no longer only duration and no longer a one-directional vector—has to be acknowledged, and indeed accounted for, in the programming. Meaning corresponds to time. Information and meaning need to be understood in their unity.

In recent years, the Boltzmann equation,

$$\mathbf{S = k \log W} \qquad \mathbf{(d)}$$

in which S stands for entropy, k is a constant (the Boltzmann constant), and the logarithm of the states of a system W ("elementary complexions")—which stands behind Shannon's work, underwent scrutiny. Tsallis [Tsallis *et al.* 2000] belongs to those who noticed that under certain circumstances, some systems will actually undergo a reduction in their entropy. Schrödinger [1944] and Elsasser [1998] were more specific: Living systems are negentropic. This new theory of disorder takes into consideration the dynamics of self-organization. Moreover, Szilard [1929], in describing biological processes, took note of the decrease in entropy in living systems. With all this in our minds, it is important that we realize that sooner or later information theory itself will have to be redefined in order for us to account for the fundamentally different dynamics of life.

Data currently processed in the various forms of computation practiced around the world is fundamentally information. In the last ten years, the focus on semantics, expressed through the interest in generating ontologies, added referential data to information. An important role in this change was assumed by those trying to develop a worldwide Web that mimics human interaction. There is a need, recognized by the entire computer science community, to make a semantic level possible: If we can associate data, as a syntactic representation, to what it represents, we might infer from information to meaning. Zadeh's [2002] computing with words (preceded by computing with perceptions) proceeds in the same direction.

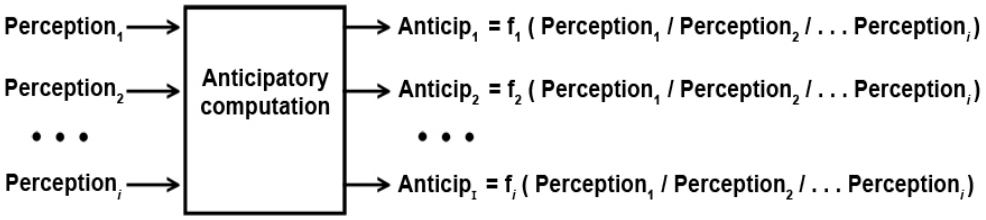


Figure 15: ***Anticipatory computing as perception-based computation***

All the new applications in automated navigation (from car parking to AI-driven automobiles), in computer-assisted surgery, in defining consumer patterns, etc., reflect the understanding that information alone is not sufficient. For me, this development justifies the hope that, some time in the future, computation will be semiotically driven.

5.2. THE BIT AND THE ANTE-BIT

Anticipation is what distinguishes the living from the non-living. Anticipatory computation [cf. Nadin 2010] can be achieved only by effectively redefining information as to include not just the semantic dimension, but, foremost, to make the pragmatics possible. In other words: A physical entity can be described in terms of information, whereas a living entity requires preoccupation with meaning, in addition to an information description. From reaction-based computation expressed in programs that are machines, to anticipatory computation, we will have to redefine many of our fundamental premises. Together with the *bit,* describing information, an *ante-bit,* describing meaning, will describe the process. The bit will effectively describe the probabilistic domain (and all the post-fact statistical information), whereas the *ante-bit* will describe the possibilistic domain (and all the pre-factual opportunities). This brings us back to the implementation of information processing in what we call computers (information processing machines, in particular, programs).

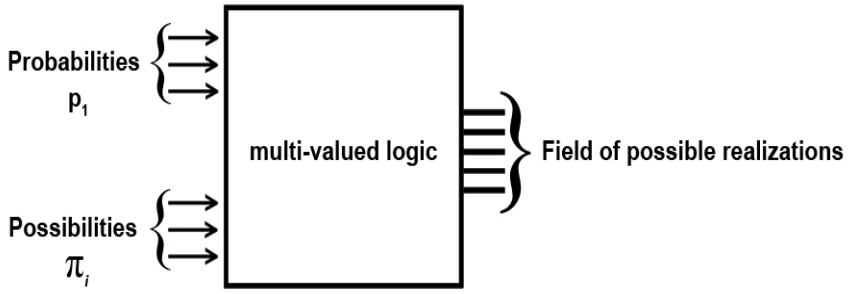


Figure 16: ***Integrating probabilities, possibilities, and multi-valued logic***

Mathematics allows us, among other things, to describe what we call *reality*. It is not the only means of description. So-called natural language can be used for the same purpose. Images, in one form or another, are also descriptions. So are sounds. Some domain-specific means of description constitute the *language* of those domains: the formalisms of chemistry (chemical formulas are a well-defined means of description), of genetics, or of logic (focused on thinking). The fact that such means of description of what there is can, at the same time, be a means of synthesizing something that is only in our minds will not preoccupy us here. But we should never ignore the complementary nature of the analytical (description) and the synthetic (design). Programs are a true exemplification of this condition.

6. DECLARATIVE AND IMPERATIVE KNOWLEDGE

A mathematical description expresses what is called declarative knowledge. One can generalize to the declarative knowledge expressed in logic, chemistry, genetics, etc. It concerns what is, as captured from a certain perspective. For instance, the ballistic equations whose solution prompted automation in a computer program describe the physics upon which artillery is based. It is well known, even by those who have never operated a cannon, that there is more to a cannon than only trajectory But for all practical purposes, the program to guide artillery actually describes the physics of throwing a ball from position A to position B. This program is no longer an expression of declarative knowledge, but of imperative knowledge: how to hit a target (which might even move from B to C as we try to target it from A). The so-called *Star Wars* defense strategy does no more than that. It could not have been conceived in the absence of computation and computer-assisted tracking procedures, doubled by a powerful inference engine.

In relation to mathematics, computer science, which has programs as a goal (among others, of course), belongs, not unlike machine engineering, to the domain of imperative knowledge. It consists of procedures that are descriptions of how to perform activities. And as any other procedure—let's say hammering a nail into a wall—it is based on recursion: It has its own actions as a reference, it is self-referential. There is an implicit circularity here: All of it is repetitive (defined in terms of what is repeated, i.e., in terms of itself). Simply stated, what governs the whole endeavor is a strategy of decomposition: Divide the action into parts. If each part has a well-defined identifiable task, this task can become a module for other procedures. The abstraction process guarantees efficiency. One does not have to reinvent the hammering of the nail each time this action becomes necessary. By the same token, recursivity speaks of the successful reduction of the task into independent procedures. This is why computers are made of machines that contain machines that contain machines, etc. Every detail is suppressed in the process. The meaning of such modules ought to be independent of parameters not essential to the task. (Remember the program called *factorial*? The factorial procedure is independent of the size of the number n. Think about a procedure returning the volume of a complex object. The parameters of the object, or the nature of its surface, or the density of the material should not affect the calculation.)

Let's be clear about the following: Declarative and imperative knowledge can be conceived only as interrelated. We can easily realize how declarative knowledge (take a mathematical equation describing the reflection of a ray of light on a mirror) is "translated" into imperative knowledge (the computer program that shows the reflection). It is by far more complicated to use imperative knowledge (a description of a scene) in order to infer declarative knowledge—a deduction: "There must be something blocking the reflection"—from it. This is an operation that is relatively often performed (for instance, in interpreting images captured by digital cameras).

There is only one reason to insist on these aspects: to make it clear that the more abstract the procedure, the more effective it is (provided that it is an adequate procedure). Abstraction ultimately means the squeezing out of time. We can build programs from modules only if they are time independent. Compound procedures have no internal time. Their reference to duration is a reference to their internal dynamics, not to the dynamics of the world. The space and time effectiveness of programs within the reductionist view of computation concerns the space (storage) and time (synchronization mechanisms) implicit in the processing, not in the entities described. However, with computation, machines open up to time through the dimension of interactivity. While every other known machine is timeless, computers open up the possibility of being driven by data from the order of events (as in playing a game), better yet, of reflecting the order of events, or of introducing (in robotics, for example) an order of events that allows for the performance of a specific task, or of achieving a complex behavior. This new dimension renders the deterministic paradigm relative. *Real-time computation has as its future the future of the process it supervises* (such as the operation of nuclear power plants or the movement of laser-guided missiles), *not the future of reality*.

In order to achieve interactivity, programs rely on mathematical and logical descriptions that reflect the dynamics of the expected activity. If you want a real-time facility for spelling correction in a word processing program, you have to provide a dynamic view of the activity called *writing.* Even on the iPad (and similar devices), writing is a pseudo-real-time interaction, supervised by spellcheckers. Obviously, the more complex the activity, the more elaborate the description and implementation in programs. For example, target recognition at microscopic levels (after labeling an ingredient of a medication, following its "moving" through the tissue subject to treatment, and even "guiding" it so that it reach a desired region), and target recognition at the interplanetary scale (the landing on Mars resulted in many examples) require detailed descriptions and elaborate program implementations. At this level, we no longer distinguish durations (the determinsitic reduction of time), but time variability: slower than real time, real time, faster than real time—as in probabilistic guessing of one's next key entry (*GoogleAssist*, for example)—variable rhythm time (slower or faster according to context). We also identify the depth of time, as in synchronicity; or as in parallel streams of time (while process 1 unfolds, process 2, related or unrelated, unfolds within the same time scale or not, etc. etc.); or as in different directions (the time vector is bi-directional, and probably even multi-directional).

7. CONSTRUCTING A NEW WORLD

Within this view, the system-based machine model has to be revisited in the sense that the clear-cut distinctions characteristic of the black box (Input, Output, States) must be ~~be~~ redefined. In particular, the local state variable describing the actual state of the computational object has to be defined in such a way that it allows for change. Computational objects with state variables that change correspond not only to trivial tasks subject to automation (such as bank account management), but also to very complex tasks (such as collaborative design over networks). Functional programming is not adequate in such cases. Imperative programming, which introduces new methods for describing and managing abstract modular

program entities, is but one of the many methods developed for this purpose. Obviously, the description (one of the many) of the living that we call *genetics* is better adapted to the task of supporting interactivity. This explains why new forms of computation emerge. They are based on the abstractions of particular fields of knowledge (e.g., genetics, quantum mechanics, DNA analysis) as we try to capture time-based phenomena. From the viewpoint of time processes, a stone is in a different situation from a living entity. With the advent of anticipatory computing [Nadin 1999] and Figure 15, this becomes more and more evident.

Max Bense [1969], the mercurial prophet of rationalist aesthetics, correctly noticed that, "It is not the mathematical description of the world that is decisive, but the principle of the constructability of the world that is gained from it." Unfortunately, as was the pattern of his activity, he did not stop in due time. He went on to speak of anticipation according to a plan of a future artificial reality and, finally, to ascertain, "Only worlds that can be anticipated are programmable, only programmable [worlds] are capable of being constructed and inhabited by human beings." The opposite is correct. Bense, like Minsky later, had the stature of an *agent provocateur*. He realized the need to address anticipation, especially in relation to aesthetic performance. In comparison, the new theoreticians of "aesthetic computation" (e.g., Fishwick [2006]) are at best pygmies, too self-important to read what others wrote long before they did.

Aesthetic computation should provide access to knowledge about the world from a perspective informed by everything that aesthetic artifacts have contributed to our understanding of reality. Chances are that aesthetic computation will prove more productive than genetic or DNA computation in defining how human beings behave. To infer from the molecular level to some disease (such as tumor formation) is, of course, a convincing knowledge path. But the same thing cannot be said about inferring from sexual behavior (*eros*) to human creativity. Evidently, aesthetic computation is based on anticipation and unites the deterministic and the non-deterministic aspects of life, and thus of interaction.

Concurrent processes, i.e., ones that take place in parallel, although not necessarily along the same time metric, are only an image of the complexity of the "time" dimension of interactive programs. The timeliness of programs that by now have adaptive qualities and display evolutionary characteristics is quite different from that of *canned* programs. The future of the latter is different from what the industry understands today when they produce the next version of a program, or of an operating system. In fact, in addressing the issue of timeliness and *future-ness* of programs, we soon come to the conclusion that technology, as the embodiment of the successful use of programs, can limit performance, but should not drive content, as still happens. As interactivity becomes the driving force, we should be able to move beyond task-based computation to a pragmatic foundation. In other words, instead of the routine of launching*canned* programs of timeliness applications (word processing, paint program, browser, etc.) under the guidance of operating systems, we should be able to execute pragmatic functions (e.g., "I want to present my data to colleagues all over the world") that would interactively select the appropriate applications and use them as we, users in a deterministic role, do today. The technology of the *apps* (prompted by characteristics of cellular telephony) is suggestive of what can be done. This role change will render the overhead of training operators obsolete. Our question regarding program timeliness will take on new meaning: Is the co-evolution of the living and the programs it conceives possible? The new computational platform should give an answer to this legitimate question.

8. PRECISION VS. EXPRESSION

Deep down, in the digital engine, there are two elements controlling and making computation possible: an "alphabet" and a "grammar." These two together make up a language: machine language. The alphabet consists of two letters (0 and 1). The grammar is the Boolean logic (slightly modified since Boole, but in essence a body of rules that make sense in the binary language of *Yes* and *No* in which our programs are

written). The assembler—with a minimum of "words" and rules for making meaningful "statements"—comes on top of this machine language; and after that, the level of *formal language* performance, in which programs are written or automatically generated. Such programs need to be evaluated, interpreted, and executed. Here I submit to the reader structural details we all know (some in more detail than others), but which only rarely preoccupy us. My purpose is very simple: *to lend meaning to my point that, in order to be meaningful, computers ought to be semiotic machines* (an idea I first articulated over 30 years ago). Too many scholars took over my formulation (with or without quotation marks or attribution) without understanding that as a statement, it is almost trivial (for example, Tanaka-Ishii [2010]). What my colleagues—some of them respectable authors and active in semiotic and computer science organizations that assert their legitimacy—have totally missed is the need to realize that such a description makes sense only if it advances our understanding of what we describe. To say that the computer is a semiotic machine means to realize that what counts in the functioning of such machines are not electrons (and in the future, light or quanta or organic matter), but *information* and *meaning* expressed in semiotic forms, in programs, in particular, or in *apps*. We take representations (which reflect the relation between what a sign represents and the way something is represented) and process them. This is the Shannon-based model. (Or should I say the Bell Telephone model?) Moreover, as we use computation, we try, after processing, to assign a meaning to our representation. Since in the machine itself, or in the program that is a machine, there is no place for a semantic dimension, we build *ontologies* (which are databases similar to encyclopedias or dictionaries) and effect association. This is how search engines frequently work; this is what stands behind the new verb "to Google" and our actions when we start research by identifying sources of knowledge in the worldwide Web.

The two-letter language (of zeros and ones) and the "grammar" (Boole's logic) allow for precision. But once we realize that we are not after information only, but after meaning as well, things get a bit more complicated. Actually, we do want to maintain precision, but we also seek expressiveness. The natural language alphabet (the 26 letters of the English Roman alphabet), along with grammar, made not only science, but also poetry, possible. Nobody in his (or her) right mind reads a poem in order to obtain information (expressed in bits and bytes), or for the sake of information. No one plays a computer game for the sake of information. *Meaning* is what the reader constructs in the interpretation or in the game action. The same holds true for interpreting the "living computation"—the meaning of change from a condition defined as "healthy" to a condition defined as "diseased." Medicine focused exclusively on information fails exactly because it ignores the meaning of changes in the information. A computational medical diagnostic has to integrate both information and meaning.

The challenge to those involved in computation-based acquisition of knowledge is to find the right combination of precision and expressiveness. Or, maybe, to develop computational means for capturing, if not simultaneously then at least subsequently, one and the other.

9. BOOLEAN LOGIC: NO ALTERNATIVE?

Even our ontologies, hand-made or automatically generated, stand on the "shoulders" of the language of zeros and ones (of *Yes* and *No*) and are captive to the Boolean algebra that is the grammar of this primitive language. Any scientist worth his or her salt should by now know that the means of representation actively influence and affect the representation. They are not neutral, but constitutive of the *interpretant processes* that make up our way of thinking. They also influence our actions. To say that computers are semiotic machines means to realize that the interpretant, i.e., infinite semiosis (the sign processes through which we become part of the signs we interpret), influences us to act differently, to think differently, to express ourselves differently from the way we did when language (and literacy; Nadin [1998]) were the dominant means of expression, communication, and signification.

The extreme precision brought about by an alphabet of two letters and a grammar of clear-cut logic comes, as we have seen, at the expense of expressiveness. The more precise we are, the less expressive the result. Fuzzy set-based descriptions are much richer in detail; three-valued logic is by many orders of magnitude more productive than the two-valued Boolean logic. Fuzzy logic is even more supportive of rich expression. In terms of program timeliness—*future-ness,* in particular—this means that we could capture time and make it a part of programs only, and only, when computation will transcend, as it partially does, not just its syntactic dimension, but also the semantic dimension of the signs making up programming languages. Indeed, at the moment when computation will be pragmatically driven by what we do, it will acquire a time dimension coherent with our own time. And it will reflect the variability of time. Interestingly enough, this is partially happening in the computation required by interactive *massively multiplayer online role playing games* (MMORPG). We are what we do, and accordingly, if we could integrate programs in what we conceive, plan, execute, and evaluate, and thus in our own self-constitution, we would establish ourselves not just as users, but also as part of the program. For this to happen, many conceptual barriers need to be overcome. We would have to address the need to redefine information—or at least to integrate in its definition our interest in how forms are generated. (The word *information* is derived from *in formare*, how we shape things in our mind.) We will have to redefine the alphabet and the logic, to rediscover quality as the necessary complement of quantity, and to understand the digital and analog together. Within my program, which is a bit more radical, this means to practice not only the deterministic reaction mode of the physical, but also the anticipatory characteristic of the living.

10. ANTICIPATION AND GAMES—A COMPUTATIONAL CHALLENGE

If you have ever played hide-and-go-seek (and who hasn't?), you don't need much of an explanation for what it means to be engaged in a game. Yet today, more people will rather recite to you the technical details of an FPS—the acronym for First Person Shooter—than recall the anticipation-driven hiding: "No, the seeker will never find me hiding behind the barrel at the cellar entrance. And if she does, I will be faster in . . . ." Shannon, as the father of information theory, made sure that at his funeral there were as many games as possible, thus reflecting his passion for them. I believe that he built the first game console [cf. Roch 2010]. I am referring to the 1953 "3Relay Kit," built with David Hagelbarger, on which 55 games, educational in nature, could be played. This happened almost 60 years ago, not in the age of Nintendo, Sony, Sega, etc. I also believe that everyone captivated by video games has failed to take advantage of what Shannon wrote about the pleasures of playing, as well as about what is needed to produce toys, such as the wonderful *Little Juggling Clowns* (see https://www2.bc.edu/~lewbel/ Shannon.html), that make a game possible. Yes, before Atari's *Pong*, there was William Higinbotham with his tennis game (1958). And before the current obsession with games—the fastest growing area in education and training—there was Shannon, examining the complexity of the human activity called *play*.

Be this as it may, Shannon would be—as we all are—impressed by the "marriage" between games and computation. It goes without saying that the majority of games taking advantage of computers remains reactive: cause-and-effect instantiations of simple (or not so simple) stories. But there are also the ambitious *massively multiplayer online games* (MMOGs, sometimes identified as MMORPGs). While hide-and-go-seek takes place in real-time-real-world (RTRW), and anticipation is associated with the living, i.e., the players, MMOG takes place in a real-time-virtual-world (RTVW), and anticipation is contributed not by the virtual characters (VC), but by the real players (RP).S

For the sake of clarity: There is a virtual world (VW), pretty much imitating the back yard for hide-and-go-seek; there are virtual characters (VC) imitating the appearance of playing children; and there is a story. As with a puppet show, behind the characters, there are real players (puppeteers). If machines

were to play the MMOG based on the rules encoded in the game, there would be no anticipation. Indeed, AI game playing is such an application: the AI procedure outsmarts the game (sometimes playing less than honestly, that is, reading the digital script). But the games I am referring to are played by real people—the majority of them young and very young—and there is real competition (sometimes for *real* money) among them. This is where the living contributes anticipation.

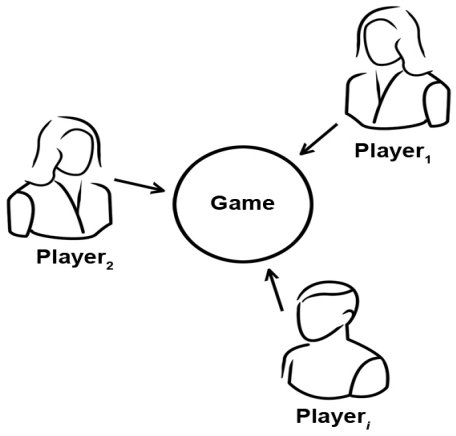


Figure 17. ***The game is the medium; the interaction among competing players involves anticipation.***

As already stated, the game is the medium, with all its implicit limitations. Competing players can communicate, but do not have to communicate. In terms of the subject pursued in this paper, we are examining computation in which the time-*lessness* of the programs making up the massively multiplayer game (3D modeling, animation, rendering, sound processing, etc.) and the players' time meet. Regardless of the type of computation involved, an MMOG, like a sophisticated interactive virtual reality application, is a medium for the expression of anticipation, without itself having any anticipatory characteristics.

Just for the sake of clarity, let us recall that what makes an MMOG possible are lots of machines embodying a large variety of mathematical descriptions. First and foremost is the mathematics of game theory (von Neumann and Morgenstern [1944]). But of no less importance is the mathematical description of the "physical" reality (levels, i.e., various stages on which the action takes place); the mathematics of interaction (among characters and the virtual world); the mathematics of persistence (you cannot start a game with some characters in a given landscape and end it with other characters in a different location).

Concept art is the blueprint for the game's world and characters: This is what the new world and its inhabitants will look like. This world is aesthetically defined. 3D modeling translates this blueprint into concrete appearances. We project textures onto the models (we "paint" them). When we need them, we cause bumps in the world through a special texture mapping procedure. There is animation, and there are particle effects (reflections of snowflakes, water, or minerals, etc.). If computationally this is not sufficiently challenging, we add the task of caching the various models and texture maps, the many accessories (often weapons and armor). Moreover, all of this has to be displayed. This entails interfaces and 3D engines for rendering the game's environment. The programming effort is monumental. Imagine generating a huge amount of machines that work in sync according to a script that is continuously changing as the living players pursue their own goals. The following diagram is only suggestive of the effort. The fact that the world of the game has to be divided into zones means that several machines are generated for providing a unified stage for the action. There is much control of actions (movement being usually recorded as a vector), collision detection (remember running into each other in hide-and-go-seek?), and a lot of location management. Of course, there is also the need to maintain and upgrade the software.

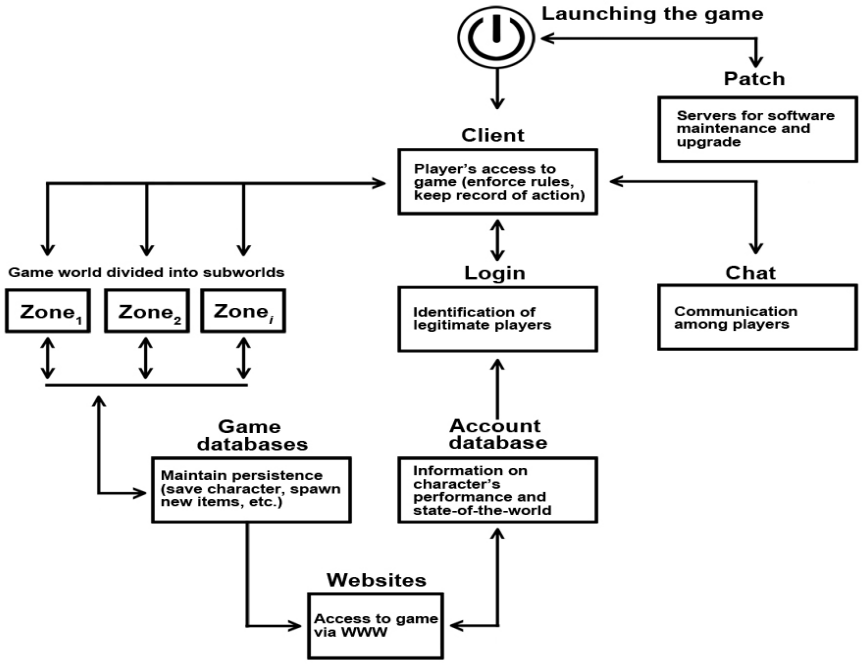


Figure 18: ***MMOG architecture***

All this effort, involving thousands of servers and millions of computers—some of them actually cell phones—where the game is played, results in a huge integrated machine, deaf and blind, unable to anticipate, although not lacking probabilistic functions (predictions of situations and actions). The machine becomes alive once it is played.

11. STUXNET—A STORY TO BE TOLD SOME OTHER TIME

Self-replication, a subject that von Neumann addressed as he tried to prove that computation can model the living, actually originated in Turing's *undecidability theorem* (1937) (cf. Sayama [2008]). In strict terms, a self-reproducing capability is described as the generation of two or more new entities similar in every respect to the initial one. It is possible that in self-replication, the original is destroyed.

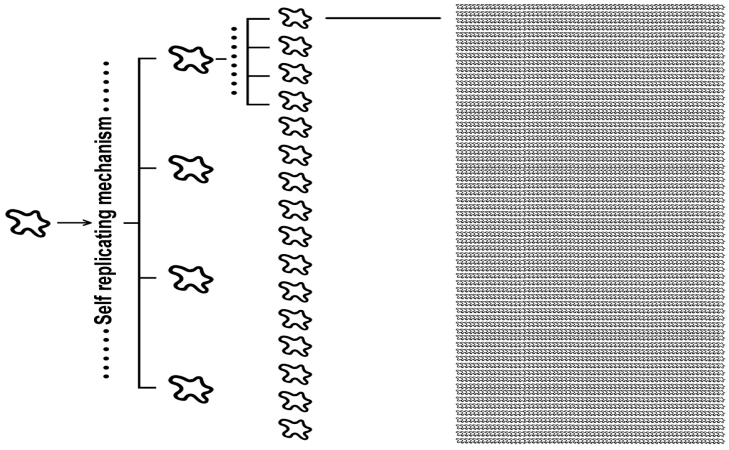


Figure 19: ***Self-replication as an open process***

Von Neumann's *Universal Constructor* is a mathematical entity, conceived before the use of computers. Like the Turing mathematical machine, it consists of a tape of cells that encode a sequence of desired actions. A writing head (called a *construction arm*) allows this conceptual device to print out (*to construct*, as it is termed) a copy of itself. There is almost no difference between this conceptual machine and the Trojan horse programs that infect our machines today. Computer viruses, which are also machines—i.e., programs built to infect other programs—perfectly embody what Minsky [1967] described as

> "the greatest advances in modern computers […] the notion that programs could be kept in the same memory with 'data' and that programs could operate on other programs, or on themselves, as though they were data."

No, Minsky was not referring to all those programs that often make our lives miserable. But he more or less made us aware that they cannot be avoided. The traditional Turing Machine model does not capture the entire behavior of viruses. More precisely, interaction and infection procedures are probably of a different computational nature.

Stuxnet has been characterized as *malware*, which will do "bad things"—to be defined—in reprogramming complicated industrial controls systems. In the final analysis, it modifies machines—the programmable logic controllers of oil pipelines and power plants, or vast electric power supply systems, gas pipelines, etc.—and makes them operate in a manner that contradicts their intended functions. The analysis of Stuxnet reveals the well-known zero-day exploits, the Windows root kit, a PLC root kit, LNK files (allowing for the execution of arbitrary code without user interaction), antivirus evasion techniques, network infection, peer-to-peer updates. The inventory of the manifestations of malware is broad. Apparently, Stuxnet can lie dormant for almost a year; it self-replicates through removable drives; spreads in a LAN through SMB; copies and executes itself through a peer-to-peer mechanism. It contacts command and control servers of variable addresses ("Update me!" is what Stuxnet, or parts of it, demands) and bypasses quite a number of security machines and programs, etc., etc. Many fictionalized variations will result from this computer reality. Security experts are like surgeons: "Show me!" (the code), after which they operate (sometimes successfully, sometimes not). The rest: a public burned by experiences with computer viruses and Trojan horse programs, and willing to accept fact and fiction. And the target*? Never acknowledge that you lost control!*

If we consider only one stream from the many "machines" integrated in Stuxnet, we have the following description as a suggestion:

$$\mathbf{state(t) = } \mathit{f}\mathbf{(state(t-1), state(t), state(t+1))} \qquad \mathbf{(e)}$$

In its current form (rather abstract), it does not inform us about the nature of the time dependency, or about the nature of the function (or aggregate of functions) describing the relational nature of the worm(s). Without suggesting an "improvement" (Who wants to cause more trouble, knowing that what we do unto our enemies will eventually come to haunt us?), I will again suggest the distributed model of intelligent multi-agent societies, in which the space of possibilities is built as a relational map (functions related to targets). If we can reduce the possibilistic space of what can go wrong, we will need fewer intelligent agents to be deployed with our applications.

Stuxnet has been the source of many reports—some directly hyperbolic—and even more attempts to prevent its actions. It is not the sensational aspect that prompts these closing lines of a study dedicated to the complexity threshold between the anticipating living and the reactive machine. Rather, it is the suggestion that important aspects of computation—such as self-replication, self-organization, creativity, etc.—hint at explanatory models for the processes through which anticipation is expressed. Imagine a Stuxnet-like entity

programmed by physicians in order to address the damaged control mechanisms of the body that lead to cancer. Or imagine self-healing technology—especially when the difference between "targets" and digital decoys (intelligent agents exercising local control and identifying attacks from outside)—that will leave a Stuxnet-like entity powerless. We can worry about what damage might result in the future as viruses, worms, and Trojan horses are deployed by less-than-friendly parties. But we can, as well, think about possible fields of application: the cure of cancer is one possible candidate; self-healing digital technology is another. It is important to understand that at the threshold of complexity that divides the living and the non-living, intervals become time, and anticipation complements reaction. The primitive anticipatory elements in Stuxnet (a product rather "rushed to market") hint at what a better understanding of anticipation can afford us.

REFERENCES AND NOTES

1. Abelson, Harold, Sussman, Gerold Jay Sussman, Julie (1996).*Structure and Interpretation of Computer Programs.* Cambridge MA: MIT Press.
2. Adleman, Leonard (1994). Molecular Computation of Solutions to Combinatorial Problems,*Science* 266 (11): 1021–1024. See also: "Molecular Computation of Solutions to Combinatorial Problems."
3. Aiken, Howard (1900–1973). By 1944, Aiken and colleagues at Harvard University, with help from IBM, developed the Mark series of computers. "The Mark I, was the world's first program-controlled calculator; an early form of a digital computer, it was controlled by both mechanical and electrical devices." See also: http://www.ideafinder.com/history/inventors/aiken.htm.
4. Babbage, Charles (1864). The Analytical Engine, *Passages from the Life of a Philosopher*, chapter VIII. (See also http://books.google.com).
5. Baran, Paul (1964). On Distributed Communications, *Rand Report*.
6. Bense, Max (1969). *Einführung in die informations theoretische Ästhetik* (Introduction to Information Theory Aesthetics). Reinbeck.
7. *Corts, Jochen and Hackmann, Daniel (2009) Risk Management and Anticipation: A case study in the steel industry. In Nadin, M. (Ed.),* Anticipation and Risk Assessment*, special issue of* Risk and Decision Analysis, *1:2, 103–111*.
8. Descartes, René (1637). *Discourse de la méthode pour bien conduire sa raison et chercher la vérité dans les sciences,* Leiden.
9. Dijkstra, Edsger (1976). *A Discipline of Programming*. Englewood Cliffs NJ: Prentice Hall.
10. Elsasser, Walter M. (1998). *Reflections on a Theory of Organisms*. Baltimore: Johns Hopkins University Press. (Originally published as *Reflections on a Theory of Organisms. Holism in Biology*, Frelighsberg, Quebec: Orbis Publishing, 1987.)
11. Einstein, Albert (1905). *Special Theory of Relativity*), or: (1916) *General Theory of Relativity* (translation 1920), *Relativity: The Special and General Theory*. New York: H. Holt and Company.
12. Feigenbaum, Edward (2003). Some challenges and grand challenges for computational intelligence, *Journal of the ACM* (JACM), 50:1, January, 32–40.
13. Feynman, Richard (1960). There's plenty of room at the bottom, *Engineering and Science*, Pasadena: California Institute of Technology. See also: http://www.zyvex.com/nanotech/feynman.html.
14. Fishwick, Paul (2006). *Aesthetic Computing*. Cambridge MA: MIT University Press.
15. Foerster, Heinz von (1976). Objects: tokens for (Eigen-)behaviours, *ASC Cybernetics Forum* 8, (3&4), 91–96.
16. *Foerster, Heinz von (1995). Cybernetics and Circularity, Anthology of Principles, Propositions, Theorems, Roadsigns, Definitions, Aphorisms, etc. See also: http://www.cybsoc.org/heinz.htm.*

17. Foerster, Heinz von (1999). *Der Anfang von Himmel und Erde hat keinen Namen* (2nd ed.). Vienna: Döcker Verlag.
18. Fraley, Dennis J. (2010). Computation is Process, *What is Computation*, Ubiquity Symposium. In Denning, P. and Wegner, P. (Eds.), ACM Digital Publication, November. See also: http://ubiquity.acm.org/article.cfm?id=1870596 (Retrieved January 10, 2011).
19. Gardner, Martin (1958). *Logic Machines and Diagrams*. New York: McGraw–Hill.
20. Greene, Brian (2004). *The Fabric of the Cosmos*. New York: Alfred A. Knopf.
21. Grier, David Alan (2005). *When Computers Were Human*. Princeton: Princeton University Press.
22. Gödel, Kurt (1931). Über formelle unentschiedbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatsh. Math. Phys.*, 38, 173–198.
23. Helmholz, Hermann von (1878). The facts in perception. In Ewald, W.B. (Ed.), *From Kant to Hilbert: A Source Book in the Foundations of Mathematics*, 2 vols. Cambridge: Oxford University Press, 1996, 698–726.
24. Hinton, G., *et al*. (1995). The "wake-sleep" algorithm for unsupervised neural networks, *Science*, 268, 1158–1161.
25. Hippocrates. 460-370 BC. For short information on the Four Humours, see: http://hippocrates.human-types.com. (Retrieved January 15, 2011).
26. Kepler, Johannes (1609). *Astronomia nova* (New Astronomy).
27. Ketner, Kenneth L (1984). The Early History of Computer Design: Charles Sanders Peirce and Marquand's Logical Machines, *The Princeton University Library Chronicle,* XLV:3, Spring.
28. Leibniz, Gottfried Wilhelm. See: *Zwei Briefe über das binäre Zahlensystem und die chinesische Philosophie*. (edited, translated, and commentary by R. Loosen and F. Vonessen). Stuttgart: Belser Verlag, 1968.
29. McCullogh, Warren S. and Pitts, Walter (1943). A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics* 5, 115–133.
30. Mettrie, Julien Offray de la (1748). *L'homme machine*. (See also, Thomson, Ann, Ed., *La Mettrie: Machine Man and Other Writings* (Cambridge Texts in the History of Philosophy), 1st Edition, 1996.
31. Minsky, Marvin L. (1967). *Computation: Finite and Infinite Machines*. Prentice–Hall Series in Automatic Computation. Englewood Cliffs NJ: Prentice–Hall.
32. Nadin, Mihai. At the Polytechnic Institute in Bucharest (1955-1960), I programmed on paper and carried out debugging on paper. It could just as well have been done in my mind.
33. Nadin, Mihai (1982). Consistency, completeness, and the meaning of sign theories: The semiotic field. In Nadin, M. (Ed), *The American Journal of Semiotics,* 1:3, 79–88.
34. Nadin, Mihai (1998). *The Civilization of Illiteracy,* Dresden: Dresden University Press. *The Civilization of Illiteracy* focuses on what makes the humankind's successive ages (referred to as pragmatic framework) possible and necessary.
35. Nadin, Mihai (1999). Anticipation—A Spooky Computation, *CASYS, International Journal of Computing Anticipatory Systems*, In Dubois, D. (Ed.). Liège: CHAOS, 6, 3–47.
36. Nadin, Mihai (2003). *Anticipatory Mechanisms for the Automobile.* Presentation at AUDI Headquarters, Ingolstadt, Germany, February 19, 2003. See also http://www.nadin.ws/archives/586 .
37. Nadin, Mihai (2010). Anticipatory Computing: for a High-Level Theory to Hybrid Computing Implementations, *International Journal of Applied Research on Informatino, Technology and Computing*, 1–27.
38. Napier, John (1617). *Rabdology* (W.F. Richardson, Trans.). Cambridge: MIT Press, 1990.
39. Neumann, John von and Morgenstern, Oscar (1944). *Theory of Games and Economic Behavior*. Princeton: Princeton University Press.
40. Peirce, Charles Sanders (1871). Charles Babbage, *Nation* 13 (9 November), 207–208, reproduced in the *Peirce Edition Project,* II, 1984, 457–459. In the same article, Peirce gave some details regarding Babbage:

About 1822, he made his first model of a calculating machine. It was a "difference engine," that is, the first few numbers of a table being supplied to it, it would go on and calculate the others successively according to the same law (p. 457).

He discovered the possibility of a new *analytical* engine to which the sufference engine was nothing; for it would do all the *arithmetical* work that that would do, but infinitely more; it would perform the most complicated *algebraical* processes, elimination, extraction of roots, integration, and it would find out for itself what operations it was necessary to perform; . . . (p. 458).

41. Peirce, Charles Sanders (1887). Logical Machines, *The American Journal of Psychology*, November, 70.
42. Peirce, Charles Sanders. *The Writings of Charles S. Peirce,* Peirce Edition Project (Houser, N. *et al.*, Eds.), Vol. 6, 1982, p. 72. Peirce, in Logical Machines (November 1887, published in *The American Journal of Psychology*) described the Jevons machine, as well as Marquand's machines. He also pointed out that the study of the transition from such machines to the Jacquard loom would "do very much for the improvement of logic."
43. Popper, Karl (1959). *The Logic of Scientific Discovery*. London: Routledge.
44. Riedel, Marc (2009). IEEE CANDE Workshop on the Future of CAD/EDA, Monterey CA.
45. Roch, Axel (2010). *Claude E. Shannon: Spielzeug, Leben und die geheime Geschichte seiner Theorie der Information* (2nd ed.). Berlin: Gegenstalt Verlag.
46. Röfer, Thomas, *et al*.(2006). *BreDoBrothers. Team Description for RoboCup 2006*. See: *citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.8094. (Retrieved January 9, 2011).*
47. Rosen, Robert (1991). *Life Itself*. New York: Columbia University Press.
48. Sayakawa, Hiroki (2008). Construction Theory, self-replication, and the halting problem,*Complexity*, 13:5, 16–22.
49. Schrödinger, Erwin (1944). *What is Life?* Oxford: Cambridge University Press.
50. Shannon, Claude E. (1940). A symbolic analysis of relay and switching circuits. Thesis (M.S.), Massachusetts Institute of Technology, Dept. of Electrical Engineering. See also http://dspace.mit.edu/handle/1721.1/11173.
51. Shannon, Claude E. (1948). A mathematical theory of communication, *Bell System Technical Journal*, Vol. 27, July, 379–423 and October,623–-656.
52. Szilard, Leo (1929). Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen, *Z. Phys.* 53, 840–856. See also: On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings, *Behavioral Science,* 9, 301–310.
53. Tanako-Ishii, Kumiko (2010). *Semiotics of Programming*. New York: Cambridge University Press.
54. Tsallis, Constantine, *et al*. (2000). Generalization to non-extensive systems of the rate of entropy increase: the case of the logistic map, *Physics Letters* A 273.
55. Turing, Alan (1950). Computing machinery and intelligence. *Mind*. 59, 433–460.
56. Zadeh, Lotfi A. (2002). From computing with numbers to computing with words – from manipulation of measurements to manipulation of perceptions, *International Journal of Applied Math and Computer Science*, 12:3, 307–324.

Images reproduced with permission from: TZI, Bremen.

Research supported by antÉ – Institute for Research in Anticipatory Systems.

The author expresses his gratitude to: Dr. Otthein Herzog, Dr. Frieder Nake, Professor Fred Turner, and Professor Charlie Gere for productive discussions. Peter J. Denning read a first version of this paper and expressed interest in its further elaboration. Elvira Nadin contributed constructive criticism of the successive versions of this text. Cassandra Emswiler contributed patience, and improved upon visuals. Heinz von Foerster would have enjoyed the opportunity to write comments. Unfortunately, he is no longer with us.