How the physical Church-Turing thesis changed the concept of machine

Gilles Dowek

The concept of machine is fundamental in informatics, but it seems to have never been completely defined. We of course know a few things about machines. We know that computers, robots, networks… are machines. We know that machines are physical objects. We know that machines, in informatics, are computing devices… But this leaves room for many concepts.

We shall discuss in this paper how the concept of machine has evolved during the last decades to become broader and broader. This evolution has been triggered by at least two factors: the development of new types of machines and a more fundamental investigation of the limits of computation, when performed by a machine.

The “original” concept of machine

A machine in general is a tool that uses a source of energy to achieve some goal. Thus, a steam engine is a machine, but a fishing net is not. In informatics, the word has a more restricted meaning, as machines are tools that use a source of energy to compute, that is to process information expressed in a language, this information processing being either the purpose of the machine, as in the case of Pascal's machine and computers, or a means to achieve other goals, as in the case of Vaucanson's automata and robots.

In this “original” definition, machines are physical systems–thus Turing machines are not machines–but physics is kept at a distance.

Indeed, Maxwell's equations are, for instance, useful to build transistors, but not to build a binary adder with transistors. In principle, we could see the adder as a physical system, where an electromagnetic field evolves over time and we could solve Maxwell's equations to explain why this system yields 10, when adding 1 and 1, but the limit conditions, reflecting the geometry of the circuit as well as its input, would be too complex for the equations to be solved. Such a circuit must instead be described at several levels of abstraction: at the level of atoms, semi-conductors, transistors, Boolean gates… so that we can understand why it yields this result.

This simultaneous use of several levels of abstractions, that is one of the pillars of computational thinking, is the reason why physics is put at a distance, as physics is involved at the lower levels– atoms, semi-conductors… –, but not the upper ones.

Moreover, these levels of abstraction have some independence: when describing an adder built with Boolean gates, the Boolean gates are made abstract and how they are built is not important, Boolean gates implemented with vacuum tubes can even be replaced with ones implemented with transistors, without affecting the high-level description of the circuit. This independence further puts physics at a distance.

Various machines

This original concept of machine has been challenged by several types of machines built–or imagined–during the history of informatics: quantum computers, relativistic computers, DNA computers, distributed computers, and analog computers.

Quantum computers. As physics is put at a distance in the construction of a machine, physical time is also put at a distance and replaced by–to use Joinet's terminology–"logical time". Indeed, the time a program takes to execute is measured in a different unit than physical time, a unit that counts the number of primitive operations executed by the program. In simple cases, the translation to physical time can be performed by dividing the logical duration by a constant: the clock frequency of the computer. But logical time is often defined more liberally, and is then more loosely related to physical time: for instance considering loading or storing of a natural number as a primitive operation, regardless the size of the number or the length of the address of the memory cell–that is, regardless the size of the memory.

This independence of logical time with respect to physical time has been challenged by the development of quantum computers. Indeed, the very motivation for quantum computers comes from a remark of Feynman that simulating a quantum evolution, on a classical computer, required more time than the evolution itself.

But, more than this comparison between logical and physical time, the idea of superposing two machines brings physics back in, as the machines to be superposed are probably not laptops with their screens, keyboards, mouse pads, usb ports… but more likely a few bits of memory materialized by the electronic states of trapped ions or, at the most, simple computing circuits. And our way to describe computers, independently of the physical nature of their memory, does not fit to describe such computers.

We can say that a computer is built from transistors. In contrast, the superposition principle is not built, it is a physical phenomenon that can simply be used as it is to compute.

Relativistic computers. More or less the same can be said of black hole computers. Here a different physical phenomenon is used: the fact that, in relativity theory, the duration of process can be different for different observers, and that, in general relativity, this duration can be infinite for one and finite for another, in particular when this observer is falling into a black hole. And, because more functions can be computed with an infinite amount of time than with a finite one–in particular, solutions to the halting problem and to the *Entscheidungsproblem* can be computed–black holes computers can compute more functions than the classical computers.

Here again, the machine computing forever outside the black hole, while the observer is falling into it, is probably not a laptop. In fact, nobody knows what such a machine could look like, but if it could be something, it would be more likely a simple physical phenomenon, as it needs to compute forever in its own coordinate system.

Here again, this dilatation of time is not something built by humans to compute, but rather a physical phenomenon used to compute.

DNA computers. DNA computers, for instance those solving the Hamiltonian path problem, are of a different kind, because DNA is not a phenomenon, but a material object, and because it is built by humans during the mitosis of their cells.

But this action is not intentional. Moreover, other animals build DNA during the mitosis of their cells and humans build DNA in the same way as the other animals do. So humans build DNA, not as humans, but as animals.

In DNA computing, DNA is therefore rather a bio-physical object used to compute, than an object built to compute.

Distributed computers. Computer networks were originally defined as machines that connect computers, the computation taking place in the nodes–the computers–and not in the edges of the network–the wires. But it was quickly understood that connecting computers allowed more computations. Not only because computation can be performed faster, but also because some algorithms–in particular, the communication protocols themselves–make sense on a network, but not on a single computer. So, the computational power is also in the network.

Swarm computing pushes this idea one step further, as swarm computers are unstructured networks of computing devices that individually have a very limited computational power. Thus, even the Turing completeness is a property that the network has, although each node does not.

The individual computing devices can either been built by humans, such as tiny computers, or not, such as ants and birds. But in both cases the network itself is hardly built, as it is unstructured.

Analog computers. Analog computers, by definition, use physical phenomena to perform some computation. But, their weak computational power and the similarity between the simulating and simulated phenomena kept analog computers at the border of informatics.

This situation has changed with the rise of Turing complete analog computers–for instance in Bournez' work. Here, it becomes possible for an analog computer to, for instance, add two numbers. And solving a differential equation describes how this analog computer yields 10, when adding 1 and 1. The analog computers relativize this process of constructing successive object–semiconductors from silicon, transistors from semiconductors, Boolean gates from transistors…–that kept physics at at a distance.

Each of these examples broadens the concept of machine as these machines are different from classical computers, built in several steps: semiconductors, transistors, Boolean gates… Indeed, in the these computers, various physical phenomena and objects are used–rather than built–to compute and physics is not put as a distance like in classical computers.

The physical Church-Turing thesis

Another factor that lead us to broaden the concept of machine is the investigation of the limits of computation, when performed by a machine: the physical Church-Turing thesis.

This thesis is simple, but it is difficult to explain, because of the tortuous historical process of its emergence. What made this process tortuous is that the first algorithm description languages–

Herbrand-Gödel equations, rewrite systems, lambda-calculus, Turing machines…–appeared before the first computers and had a purpose very different from that of programming them.

So, before the first computers were built, a first thesis–the original Church-Turing thesis–was formulated as the fact that no algorithm description language can express an algorithm solving the halting problem, or any other algorithm computing a non recursive function.

After the first computers were built, a second thesis–the physical Church-Turing thesis–was formulated as the fact that no machine can execute an algorithm solving the halting problem, or any other algorithm computing a non recursive function.

In both cases, the formulation of this thesis did not just required to apply the predicate "algorithm description language" or "machine" to a constant, as in the statements "the lambda-calculus is an algorithm description language" or "the Small-Scale Experimental Machine is a machine", but to apply it to a quantified variable to form the statements "no algorithm description language can to express an algorithm that...", "no machine can execute an algorithm that..." and this requires a more precise definition of the concept of "algorithm description language" and "machine".

Such a precise definition of the concept of machine has been given by Robin Gandy, to prove the physical Church-Turing thesis.

Gandy's definition is still somewhat conservative as he describes a machine as a mechanical built device. But the properties that such a mechanical device must verify–the "principles of mechanism"–paved the way for an evolution of this concept of machine.

In particular, Gandy stated a principle according to which, in a machine, an element of a bounded size has a bounded state space, that is that information has a bounded density. And this hypothesis is central in the proof that a machine then cannot compute non recursive functions. This "principle of mechanism" can be seen as a part of the definition of the concept of machine: in some physical systems in which information has a bounded density and in others not, and only the former can be called "machines".

But, roughly at the same time, Jacob Bekenstein developed the idea that information has a bounded density in all physical systems–the bound being the inverse of the Planck constant expressed in an adequate unit system. So, all physical systems happened *a posteriori* to be "machines" in the sense of Gandy's definition.

The contemporary concept of machine

Both the development of new types of machines, such as quantum computers, relativistic computers… and the investigation on the limits of computation led us to broaden the concept of machine, from a clock-like device, built with gears and springs, to, almost, any physical system.

This has led us to investigate not only how a machine could be built with gears, vacuum tubes, transistors… but also which physical phenomena and objects could be used to compute: the superposition principle, the dilatation of time, the duplication of DNA, the distribution of matter in space… This has led us to relativize the need to put physics at a distance. This has also led us to investigate how a physical system could evolve: the physical Church-Turing thesis forbids any physical system to turn green or red according to the termination of a Turing machine, encoded in one way or another, in its initial state. This has, in turn, led us to relativize the difference between object built by humans and those that are not, just like the discovery, in Oklo, of nuclear reactor, where self-sustaining nuclear chain reactions have occurred 1.7 billion years ago.

So can we conclude that a machine is just any physical system? Not quite, because something would still be missing in such a definition.

Indeed, consider a body freely falling in vacuum. By choosing an adequate unit system, the distance traversed by this body in a duration t is $y = t^2$. But does this machine compute the square or the square root? The answer depends on the interaction protocol we define with this physical system. If we choose the time and measure the distance, the machine computes the square function. If we chose the distance and measure the time, it computes the square root.

So, a machine is not just a physical system, but a physical system equipped with a protocol interpreting the evolution of the system. The computation is not in the physical system alone, but also the way the physical system is used to compute.