Towards the Epistemic Transparency of Algorithms

Extended abstract for a talk at
Tacit Engagement in the Digital Age and Polanyi Society Meeting
Cambridge CRASSH 27, June 2019

Mihály Héder

Summary

If we define computing as the typical activity of computers – in contrast to other approaches that define computation as an abstract mathematical process – it will become evident that in some cases the result of the computation is not only a function of the algorithm executed but also of certain physical processes. For example, random numbers, which are used in many algorithms, are usually generated by sampling the physical environment of the computer. In other words, we should not forget that computing is embodied, and different embodiments lead to different computation results. This has far-reaching security and transparency implications, especially in the case of machine learning applications like evolutionary computation.

1 Introduction

In the last one or two decades, engineers trying to solve hard computational problems invented a solution that they call *emergent computing*. This development is not based on a dialog between philosophers (who have known the concept of emergence for some centuries now) and engineers; it is rather the result of the internal evolution of computer science, independent of the philosophical debates.

We have to consider two different approaches to the problem of emergence, with independent roots, then. But this does not mean that emergent computing and the philosophical notion of emergence are incompatible.

On the contrary, there is some literature already written discussing how emergent computing fits into the original philosophical debate. These works mostly conclude that computing can only be *epistemically* emergent (e.g., Cariani 1991), not *ontologically*. For most of the authors, the epistemic nature of computing means that for an observer, the computation produces a surprising, complex phenomenon. But the phenomenon is identical with a set of more primitive building blocks, which are manipulated by simple operations. So the complex phenomenon is not a novel entity existing on its own, no matter what the impression of the observer is.

This view is the best summarized by Barry S. Cooper (2011). He describes emergent computing like this: 1) An engineer creates a computation, using a program on level L1. 2) An observer recognizes a complex phenomenon, or behavior, like the typical complex dynamics of a stock market. Cooper calls this an L2 level observation. 3) For the observer, it is not evident how the instructions of the L1 program lead to the phenomenon on L2. What is more, as many articles on emergent computing applications have reported, sometimes the connection is not immediately apparent to even the programmers themselves. Nonetheless, in this line of thought, an L2 phenomenon is nothing more than an arrangement of elements on L1, hence we can only say that L2 phenomena emerge in an *epistemic* sense.

I will argue that a proper model of computers and their computation is crucial. Replacing our standard view of computers with a more accurate one enables a richer and more exciting interpretation of emergent computing. I will show that a computer at a given point in time cannot be fully described by the contents of its memory and the blueprint of the hardware. We also have to take into account its current physical state and its environment's physical state, because these factors also figure into its next state. This does not mean that one could not reduce a complex phenomenon to the bits of a given snapshot of a computer's internal state in a single point of time.

2 Embodied computation

Let us start with the recognition that every computation that has been carried out is embodied somehow, either in a living organism or in a machine. The next question is if one calculates the computation 2 + 2 in one's head, and then by pressing "2", "+", "2" on a calculator, are the two computations identical or not? If they can be said to be identical, what exactly creates the connection between them? In the philosophy of mathematics, it is well known that many mathematicians think that there are abstract mathematical calculations that are governed by abstract principles. The actual calculations are implementing the abstract 2 + 2 in one way or another. This is the Platonist understanding of mathematics in a very small nutshell (for more, see James Robert Brown 2005, chap 2). Of course, the same abstract calculation can be realized multiple times, and the result is always the same if the realization is done right (of course, other properties of the computation, e.g., its speed, might vary). As a corollary, the mathematical analysis always describes the abstract calculation or computation, and not physical realization.

The connection between an abstract calculation and its physical realization is not always as close as in the case of simple addition: for example, a stable Markov chain (a common tool in computing theory for describing certain stochastic processes that are characterizable by discrete states, and in which future states are only dependant on current states and not on past ones – that is the Markovian property) cannot predict the specific state of the process at a future point in time, only the possibilities of certain states.

Following this train of thought, one quickly recognizes that the examples of computer applications are very different from each other in the sense of mathematical definability, even though all of them are computer applications.

The problem of embodied computing – like the problem of emergence – was explored by computer scientists independently from the philosophical discussion of embodiment. One reason that scientists' attention turned to embodied computing was that certain problems appeared that endangered the sustainability of the fast pace of the evolution of computing hardware in recent decades. As MacLennan (2008) describes the situation, while earlier every bit was represented by a very large number of physical particles, in the latest systems, this ratio has started to decrease. Systems in which the number of bits and the number of particles that store the bits are increasingly close have become a real prospect shortly. For researchers working on a scale this small, and tackling the properties of matter that allow the storage and manipulation of more bits, the idea of different computer architectures quickly surfaced.

The core of the idea is that strictly adhering to the Church-Turing computational principles is not always rational, especially if these principles do not fit the capabilities of the matter we want the computer to be made of. Maybe it is more rational to make computers that do not qualify as Church-Turing implementations but are quicker or more energy efficient in return.

One approach is to tolerate a certain rate of error in the computation to gain dramatic improvements in speed. Certain applications do not need perfectly accurate calculations anyway (for example, in a fast-paced flight simulator, who would notice tiny errors in the movement of the water in the ocean below the plane?), and in other applications error prevention can be added at higher levels of the architecture (IEEE Spectrum, 2009).[1] Quantum computing also exploits the nature of matter to go beyond the Church-Turing computation.

3 Evolutionary Computing

There are many applications that run evolutionary computations and genetic algorithms. One such application was Tierra, created by the biologist Thomas Ray. He defined a virtual computer architecture that used a 5-bit instruction set, meaning that there were 32 different instructions.

For this architecture, he wrote a program that was able to copy itself repeatedly to different memory locations. This way, the copies of the small program slowly consumed the memory space available. However, sometimes the system modified a random bit during its operation. This represents a mutation. Most of the time, the modified bit makes the program disfunctional. But sometimes the modified code is functional, or even more viable than the original (e.g., the program is made of fewer instructions). In other words, a process of artificial evolution starts (Ray 1991). A similar experiment is Nanopond[2], which uses a 4-bit instruction set and contains no initial program – the initial program itself is created by modifying random bits in huge memory space.[3]

So the next question concerns the generation of random numbers. There are many methods to solve this problem. One approach is using Pseudo-Random Number Generators (PRNG).

These are mathematically fully defined algorithms, which, using a certain set of initial parameters, can give numbers that approximate the properties of good random numbers: that is, they do not start to give the same numbers again for a long number of generations, and the generated numbers are nicely distributed between a minimum and the maximum value, instead of forming a tight pack around a certain number.

PRNGs have many problems. The biggest one is that with certain (a priori unknown) initialization parameters (commonly called as seeds), the period in which the random numbers do not repeat can be surprisingly short. With other seeds, the distribution is uneven. Because of these problems, no current mainstream operating system uses PRNGs on their own. Instead, they use input from the environment of the computer to generate random numbers: the key presses of the user; the network interrupts that are generated by the user and the others communicating with her, e.g., in chat; certain properties of the physical layer of the network (e.g., the variations in strength of the wifi signals). Sometimes the spinning characteristics of the hard drive are used, which depend on the temperature and the position of the device; various temperature gauges that measure the processors, the discs, or the temperature of the palm rest can also be used; audio and video input devices or even the user's file names or creation times

[1] http://spectrum.ieee.org/semiconductors/processors/cpu-heal-thyself/4
[2] http://adam.ierymenko.name/nanopond.shtml
[3] For an 8 hour long Nanopond run, see: http://video.google.com/videoplay?docid=4775386042852459808

might be used. These inputs fill up an “entropy pool” in the system. Of course, the distribution of such input data can be uneven, but that can be solved in algorithmic ways, e.g., by PRNGs.

As a consequence, we know that Thomas Ray’s interesting results were affected by the environment of his computer, just like the results of the Nanopond experiments.

We can conclude that the result of running an algorithm involving randomness is a product of many physical processes. Some of these processes are just the realizations of well-defined algorithms, and the particular manner of the physical realization does not affect the result. Other processes involved appear to be completely outside the realm of mathematics. These might very well be perfectly deterministic processes on their own, but they should be in a *random relation* to the computer that executes the computation (for the definition of randomness used here see Paksi 2012).

What is more, these processes are selected exactly on the basis that they are hard to describe mathematically and are contingent on the computer’s internal state. This makes it impossible to predict the random numbers, which increases security.

To summarize, the result of the evolutionary computation is a product of many processes, and only a few of them are fully algorithmically described.

4 Machine Learning and Epistemic Transparency

Following this line of thought, we can recognize that not only the special case of evolutionary computation behaves like this, but most of machine learning.

However, the received view of computing does not include its embodied nature and is dominated by the image of an algorithm executed by a Turing machine.

Why is it so important to discuss the embodied nature of computation before analyzing emergent computing? Because, in the received view of computing, computations are mostly treated as algorithm realizations, and they are thus analyzed using algorithm theory.

Let us return to the argument discussed in the first chapter: that the central element of emergent computing is surprise, which results when scientists create a computation program on level L1, and an observer recognizes a complex phenomenon on a higher level, L2. However, the argument continues, the observation on L2 is nothing more than a set of the elements of L1, which are in turn determined by any given state of the program.

But the example of the evolutionary computing shows that the result cannot be derived from the algorithm itself. We can only say that we see the product of an algorithm *and* other physical processes.

Similarly, it is not true that a given state of computation is always reducible to the previous state and through a finite number of steps to the initial state. Also, a given state of a computation is not deducible from the initial state itself.

If we want to explain the current state of computation in general, then we have to record and take into account all of the input from physical processes since the initial state. And what we will see is that certain physical processes independent of the algorithm push the computation into a certain direction.

Another property that is often mentioned in connection to irreducibility is unpredictability. If we consider a computation that takes only initial parameters and then works inputless, we will find that every state is deducible from the initial state; as a corollary, every state is predictable – by a faster execution of the same program on a different computer. This also means that the computation in question can have multiple realizations.

This is not the case with evolutionary computing. Predicting the state of a given future step would involve not only calculating the algorithm faster but also predicting the states of the physical processes that figure in the computing. That is, the processes used in a random number. However, it was our intention in the first place that the processes in question should be unpredictable.

This does not mean merely that prediction is impossible due to some practical epistemic limitation. Neither the algorithm nor the external physical processes determine the outcome of the computation, and their relation is random. Unpredictability is only a symptom of indeterminacy in this case. And as long as predictability and transparency count as a security-enhancing features, the very nature of machine learning is fundamentally at odds with security. This is because the whole point of machine learning is that the engineer can avoid the complete exact pre-programming of the system for all possible situations – which is both impossible and undesirable as it prevents the future flexibility of the system. Therefore, a step towards transparency of algorithm is to separate the design and learning phase from the production phase, and in between the outcome of the learning process may be analysed.

References

Brown, J. R. (2005). Philosophy of Mathematics: an Introduction to a World of Proofs and Pictures. Routledge.

Cariani, P. (1991). Adaptivity and emergence in organisms and devices. World Futures: Journal of General Evolution, 32(2-3), 111-132.

Cooper, B. S. (2011). From Descartes to Turing: The Computational Content Of Supervenience. In *Information and Computation* (editors Mark Burgin and Gordana Dodig-Crnkovic), World Scientific Publishing Co., 2011, pp.107-148.

MacLennan, B. J. (2008). Aspects of embodied computing. Tech. rep., Technical Report UTCS-08-610, University of Tennessee.

Paksi D. 2012. The Meaning of Randomness. Manuscript, submitted to Tradition and Discovery.

Ray, T. S. (1991). Documentation for the Tierra Simulator. Tierra Simulator, 4.