ACADEMIA | Letters

Effective Computing Revisited

Jaime Seguel, Department of Computer Science and Engineering, College of Engineering, University of Puerto Rico at Mayaguez

---

Effective computation refers to a calculation method that demands no insight, intuition, or ingenuity for producing a result. This is, a step-by-step recipe executable by an automaton, or, in a broad sense, an algorithm. But since algorithm is nowadays so tightly tied with electronic computing, effective computation fits better with the content of this note.

The search for effectiveness reached its peak at the beginning of the twentieth century when faced with contradictions, inconsistencies, and paradoxes, some of the finest mathematicians of that time focused their attention on the foundations of the discipline. The three volumes of *Principia Mathematica* written by Bertrand Russel and Alfred N. Whitehead between 1910 and 1913, was, according to Russell, an attempt to "*show that all pure mathematics follows from purely logical premises and uses only concepts definable in logical terms.*" For Russel and Whitehead consistency was a consequence of the selection of the right logic postulates. David Hilbert went a step further. In 1921 he presented the so-called Hilbert's program, which aimed at embedding all mathematics in a formal axiom-system endowed with a precise language and manipulation rules. That system should also exhibit the next three fundamental properties.

a) *Consistency*, that is no contradiction would arise within the formalism.

b) *Completeness,* meaning that all true mathematical statements could be proved within the formalism.

c) *Decidability*, meaning that the truth value of a statement could be decided with an *effective method*.

Property c) implied that syntax not meaning was the crucial thing in a mathematical proof. For good or ill, Kurt Gödel proved in 1931, in two landmark theorems, that Hilbert's program

Corresponding Author: Jaime Seguel, seguel.jaime@gmail.com
Citation: Seguel, J. (2021). Effective Computing Revisited. *Academia Letters*, Article 3466.
https://doi.org/10.20935/AL3466.

was impossible. Gödel's first theorem showed that in a consistent formal axiomatic system that is endowed with at least the expressive power of Peano's arithmetic, there will always be statements that can neither be proved nor disproved. Those systems are thus, incomplete. Gödel's second theorem showed that no formal systems of this kind of expressive power can prove its own consistency. Therefore, a) was impossible too. Gödel's didn't address property c) because, by then, the concept of effective computation was still intuitive. But that changed in 1936 when Alonzo Church and Alan Turing, working independently, produced mathematical definitions of effectively computable functions. Church statement declared a function computable if and only if it was recursively enumerable. This characterization, however, did not quite convince Gödel. Turing's a-machine, later called Turing machine, was more promptly accepted. This abstract machine is a mathematical formalism that executes all basic mechanical actions of a human computer, this is read, write, and move from one step to the next according to a table of instructions. Turing demonstrated that, despite its simplicity, his machine could execute any problem-solving procedure described as a finite sequence of steps. In 1937, Turing also showed the equivalence of his machine to Church's characterization of computable functions, giving rise to the Church-Turing thesis: "*All models of effective computationare equivalent to, or weaker than, Turing machines*." The advent of Turing machines and the Church-Turing thesis launched theory of computation as a new mathematical discipline.

It is important to note that, unlike what is said in property c), a Turing machine does not prove a statement, but answers a question about it. This is, given a statement P on a variable x, say $P(x)$, and a specific value of x, say a; a Turing machine answers mechanically whether $P(a)$ is true. If a Turing machine can answer this question for each a in the domain of discourse of the variable x, we say that problem P is *decidable*, and call it undecidable if no Turing machine can do that. But the decidability of problem P is not the same as the provability of statement P. To prove $P(x)$ with a Turing machine we need to answer $P(a)$ for each a in the domain, which is impossible if the domain is infinite, as are most domains of interest. Perhaps the best contrast between provability and decidability is given by the statement of the halting problem: "there is no effective procedure to determine if an algorithm will halt on a given input." This statement is provable and was proved by Martin Davis[1]. But what the statement says is precisely, that the problem of testing whether a computer program will halt on an input, is undecidable.

Indeed, effective computation is applicable to a relatively small set of problems. And only for that small set of problems, the solutions depend on syntax, not meaning. It has been proved that that syntax has the structure of a context-sensitive grammar or weaker. This detachment of meaning and interpretation is a reason behind the wide applicability of computation. How-

Corresponding Author: Jaime Seguel, seguel.jaime@gmail.com
Citation: Seguel, J. (2021). Effective Computing Revisited. *Academia Letters*, Article 3466.
https://doi.org/10.20935/AL3466.

ever, meaning and interpretation are still fundamental in finding, designing, and applying a Turing machine to a problem. For example, and this is just one example, the article by Davide Castelvecchi[2] reports on an experiment to compute the spectral gap, which is the energy leap between the ground and the first excited state of a system. In the experiment, the quantum states of the atoms in a 2-D lattice are interpreted as steps in the computation of a Turing machine. The article reads: "*Cubitt and his colleagues showed that for an infinite lattice, it is impossible to know whether the computation ends, so that the question of whether the gap exists remains undecidable."*While this argument does not a prove the problem undecidable, it illustrates one of many natural phenomena that are interpretable as a Turing machine. However, there is one instance where the word interpretation falls short.

In 1957, years after the advent of the Church-Turing thesis, Francis Crick introduced the dogma of macromolecular biology. The dogma is the process of transcription from DNA to proteins. More than being interpretable as a Turing machine, the dogma is itself an effective biochemical computation method that has striking similarities to Turing machines. At the core of the dogma, DNA strings are read by RNA and then, are biochemically written as protein strings, within the cell. David Searls[3] showed that the syntax of the DNA strings that are operable by the dogma has the structure of a string variable grammar (SVG). SVG is weaker than context-sensitive grammar, and therefore, these DNA strings are also operable by Turing machines.

Transcription is what makes the cell live, and life forms first appeared on Earth around four billion of years ago. So, effective computation emerged in nature long before mathematicians thought about Turing machines. Jean Paul Sartre's assertion that *existence precedes essence* seems applicable here. Indeed, the Turing machine is the result of distilling meanings and interpretations down to the essence of what effective computation is. There is no reason to think that the emergence of the dogma has any parallel to the creation of the Turing machine. There does not seem to be in nature anything analogous to interpretation or meaning, nor to the concepts of demonstrable statement, incomplete system, or undecidable problem. Understanding the forces and mechanisms behind the emergence of the dogma can add important elements to the understanding of the ontology and epistemology of emergent systems. So far, most research in these subjects center around emerging phenomena in physical or social sciences, and is essentially descriptive in nature. I have the impression that the constructive approach by Alberto Pascual-Garcia[4] could be more adequate for studying the emergence of effective computation in nature.

Corresponding Author: Jaime Seguel, seguel.jaime@gmail.com
Citation: Seguel, J. (2021). Effective Computing Revisited. *Academia Letters*, Article 3466.
https://doi.org/10.20935/AL3466.

References

[1] Davis M. D. (1956) A Note on Universal Turing Machines, Princeton University Press, pp. 167-175.

[2] Castelvecchi, D. (2015) Paradox at the heart of mathematics makes physics problem unanswerable. *Nature* (2015). https://doi.org/10.1038/nature.2015.18983

[3] Searls D,B. (1995) String variable grammar: A logic formalism for DNA languages. The Journal of Logic Programming, Elsevier, pp. 73-102.

[4] Pascual-García A. (2018) A constructive approach to the epistemological problem of emergence in complex systems. PLoS ONE 13(10): e0206489. https://doi.org/10.1371/journal.pone.0206489

Corresponding Author: Jaime Seguel, seguel.jaime@gmail.com
Citation: Seguel, J. (2021). Effective Computing Revisited. *Academia Letters*, Article 3466.
`https://doi.org/10.20935/AL3466`.