Review

Matteo Plebani* and Luca San Mauro

Computability theory as a *philosophical* achievement

https://doi.org/10.1515/cclm-2022-0710
Received July 25, 2022; accepted July 26, 2022;
published online August 15, 2022

Abstract: Artificial intelligence plays an important role in contemporary medicine. In this short note, we emphasize that philosophy played a role in the development of artificial intelligence. We argue that research in computability theory, the theoretical foundation of modern computer science, was motivated by a philosophical question: can we characterize precisely the class of problems that can be solved by algorithms? We suggest that reflecting on the connection between philosophy and artificial intelligence helps realize that philosophical and scientific progress are connected.

Keywords: computability theory; conceptual engineering; Rice's theorem.

Introduction

To say that artificial intelligence is a scientific and technological achievement sounds like a platitude. In this note, we want to draw attention to a less obvious way in which artificial intelligence is a success story. We will argue that computability theory, the theoretical underpinning of artificial intelligence, is a significant *philosophical* enterprise and a prime example of *successful* philosophical work. In "The philosophical significance of computability theory" Section, we explain in which sense the work of the founding figures of computability theory can be viewed as an example of successful philosophical work. "Computability theory and the philosophy of language" Section discusses one case study where results in computability theory can be philosophically illuminating: Frege's distinction between the sense of a linguistic expression and its reference.

The philosophical significance of computability theory

Philosophers deal in concepts. They aim to deepen our understanding of the concept of justice, freedom, causation, rationality, and so on. Computability theory is philosophically significant because it provides a precise mathematical understanding of the concept of *solvability*, which corresponds to the (abstract) potential of solving a certain problem by algorithmic means. Algorithms are ubiquitous. When we learned to do additions, multiplications, subtractions, and divisions, we also learned a bunch of algorithms. These are step-by-step procedures to solve a problem, where every step consists of a purely mechanical operation, like writing a symbol, pressing a button, or moving a cursor to a definite location. A broader definition of an algorithm is a collection of instructions which enable to fulfill a certain task; hence, cooking recipes, medical protocols, or driving directions all qualify as examples of algorithms.

At first glance, it may seem that the problem of whether a given task is (algorithmically) solvable is highly contextual. So much so that, in most situations, the solution to a solvability question would ultimately depend on the toolkit that one is permitted to employ. For example, while one needs apples and appropriate cooking tools to cook an apple pie, considerably more sophisticated technology is required to complete a space journey. Similar considerations apply to mathematical tasks. On the one hand, it is child play to multiply two given positive integers, and any diligent high school student knows how to solve any given quadratic equation. On the other hand, however, it is well known that.

(1) If one is only allowed to use a compass and a straightedge, it is impossible to construct a square with the same area as a given circle.
(2) There is no general formula for solving polynomial equations of degree greater or equal than 5 which uses *only* addition, subtraction, multiplication, division, raising to integer powers, and the extraction of nth roots.

***Corresponding author: Matteo Plebani**, Filosofia e Scienze dell'Educazione, Universita degli Studi di Torino, Torino, Italy, E-mail: matteo.plebani@unito.it. https://orcid.org/0000-0002-4853-7865
Luca San Mauro, Universita degli Studi di Roma La Sapienza Dipartimento di Matematica Guido Castelnuovo, Roma, Italy, E-mail: luca.sanmauro@uniroma1.it

Therefore, it would seem that the search for algorithms will never end because whether a task can be solved or not largely depends on which methods are allowed; any job that now appears to be insurmountable may one day be solved with the right combination of intellect and technology. The startling realization that this is not the case gave birth to computability theory in the 1930s. Alan Turing, Alonso Church, Stephen Kleene, Kurt Gödel, Emil Post, and others demonstrated that there is a robust and mathematically precise concept of solvability, which encompasses all conceivable algorithms. (For a detailed historical reconstruction of the birth of computability and its development, see [1, 2]). A major evidence of this fact is that a plethora of distinct models of computation – such as Turing machines, register machines, partial recursive functions, λ-calculus, flow diagrams, and more – are all equivalent, in the sense that they provide the same answer to the question: "Which arithmetical functions can be computed by an algorithm, and which cannot?". As Gödel noted [3, p. 84]: "With this concept [computability] one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion". It is impossible to overestimate the importance of such an achievement. For one, it launched the Digital Age we live in, as everyday computers are nothing else that the physical manifestations of Turing machines, the simple devices – and yet as powerful as they could be (see below) – first introduced in [4].

Why is this important to philosophy, one could wonder. Let's talk about a few reasons. First, computability theory opens the door to the possibility of showing that certain problems are *absolutely unsolvable*. Note that, without a precise characterization of solvability, it would be impossible to prove that a problem is unsolvable by any algorithm. This is because the failure of finding an algorithm could always be explained with other factors, such as the lack of ingenuity, rather than by the fact no such algorithm exists.

As is clear, knowing that a task is unsolvable allows to save time and energy: once we know that a problem cannot be solved algorithmically, looking for an algorithm to solve it would be as pointless as looking for a pair of positive integers m, n such that $(m/n)^2$ is equal to 2.

The quintessential example of such absolute unsolvability is given by the Halting Problem, i.e., the decision problem which, for any arbitrary algorithm $\boldsymbol{A}$ and input x, asks whether $\boldsymbol{A}$ halts on x, or runs forever. So, there is no algorithm which is able to successfully predict for any other algorithm if it will converge or diverge on a given input. As a matter of fact, unsolvability arise in many different (mathematical) areas: if a mathematical system allows to store and process information in sufficiently complicated ways, then it will be computationally universal, and therefore even simple questions about these systems will be unsolvable. Examples of unsolvability occur even outside mathematics [5]. (For the record, computability theory doesn't stop after discovering that some problems are unsolvable. Far from it. In fact, a major thread of research within the theory has been to develop robust hierarchies of unsolvability which allow one to measure just how much unsolvable are certain problems. In a nutshell, one says that a problem $\mathcal{P}_1$ is harder than a problem $\mathcal{P}_2$ when there is an algorithm that converts any solution to $\mathcal{P}_1$ into a solution to $\mathcal{P}_2$. For a rich presentation of several hierarchies of unsolvability, see the textbook [6]).

So, unsolvable problems escape the capabilities of even the most powerful computers of today. In fact, we say with mathematical confidence that no future computer can ever solve such tasks: anything that cannot be solved by a Turing machine will always be outside the scope of algorithmic solvability. Behold how strong these statements are, and how they subvert a naive epistemological prejudice. Indeed, it would be natural to assume that a claim of the form

No computer will ever be able to accomplish the task P (1)

would be essentially empirical and open to confirmation or denial depending on, e.g., future hardware/software advancements. Contrarily, according to computability theory, one can confidently justify numerous instances of (1) by completely ignoring empirical features.

Another reason why having a mathematically precise characterization of solvability is an important philosophical achievement is that it would allow us to explain the notion of problem that can be solved by an algorithm to an alien intelligence capable of understanding basic arithmetical notions, but without any experience of our paper-and-pencil procedures to compute mathematical operations, or of anything like our usual daily practices with algorithms [7]. imagines an alien intelligence who understands basic arithmetical concepts by direct acquaintance rather then by an algorithm. Such an alien doesn't compute the result of a sum – it sees the result. It might be controversial whether understanding addition without relying on any algorithm is really possible. However, the point is that even if we allow for such a possibility, we might explain the distinction between the arithmetical functions that can be computed using an algorithm and those that cannot to a being who grasps basic arithmetical notions by direct acquaintance, i.e., in a way that is completely different from the way we understand those notions.

Remarkably, the precise characterization of the notion of solvability is the output of *conceptual* work. For example, Turing came up with the definition of Turing

machines by stripping inessentials from the process of computation as experienced by an (idealized) human being. Philosophers debate whether this work on concepts is best seen as an analysis revealing what those concepts are or rather as a sharpening of our pre-theoretical notions that makes them more precise than they originally were [8, 9]. We can remain neutral on the issue of whether the precise characterization of solvability offered by computability theory should be seen as an analysis or a sharpening. The point is that the work of Turing and others shows that putting our own concepts under philosophical lenses can have a huge impact, both on the theoretical and at the practical level.

Finally, showing that there are tasks that algorithms cannot perform raised the question whether the human mind's capacities exceeds those of a computer. Since the (in)famous [10], the issue has been a controversial topic. It is fair to say that the arguments of those arguing for the superiority of the human mind over machines has been generally received as flawed or unconvincing (see [11, 12] for a survey of the relevant literature.) However, one might still interpret the results of computability theory as supporting a disjunctive claim [13]: either the human mind can solve problems that no machine can solve, or there are problems that the human mind cannot solve. Hence, it is safe to say that computability theory provided a model to which the human mind can be fruitfully compared and that learning about the limits of computation prompted the inquiry about the limits of human knowledge and understanding.

Computability theory and the philosophy of language

In this section, we will highlight that the kind of conceptual clarity that computability theory brought on can be fruitfully applied also to philosophical questions apparently unrelated to the subject of computation.

"How do computers work?" and "How do words refer to things out there in the world?" might strike many as two completely unrelated questions, the first belonging to the field of engineering and the second to the philosophy of language. Computability theory provides a way to connect them.

Let us start with the question of how words refer to things. The philosopher and mathematician Gottlob Frege introduced a distinction that has become standard in the philosophy of language, the distinction between the *sense* of a linguistic expression and its *referent*. The two numerical expressions "$7 + 1$" and "4×2" denote the same number, eight, and hence have the same referent in Frege's terminology, but they have different senses, because they present the number eight in two different ways: as the sum of 7 and 1 and as the product of 4 and 2, respectively.

Frege did not offer a precise definition of the notion of sense. He speaks of the sense of an expression that refers to an object as a way of presenting that object. The idea is that two expressions might stand for the same object but characterize/describe it in different ways. A standard non-mathematical example is the pair of expressions "the morning star" and "the evening star", which both refer to the same celestial body, but present it in two different ways.

There is a passage in Frege's classic paper [14] where he says that:

> The sense of a proper name is grasped by everybody who is sufficiently familiar with the language or totality of designations to which it belongs; but this serves to illuminate only a single aspect of the referent, supposing it to exist. Comprehensive knowledge of the referent would require us to be able to say immediately whether every given sense belongs to it. To such knowledge we never attain [14, pp. 210–1]

This can be paraphrased by saying that there is no algorithm to determine whether two given senses are associated to the same object: an extremely interesting philosophical claim. However, it looks impossible to establish it or refute it without a precise definition sense. Computability theory can be used to offer a precise reformulation of Frege's claim, a reformulation that can actually be proved.

Let 'f' and 'g' be two symbols that stand for numerical functions. Let the sense of 'f' be the algorithm associated with 'f', i.e., the procedure that we follow to calculate $f(n)$ given n. The reference of 'f', on the other hand, is a set of ordered pairs of natural numbers, where (m, n) belongs to such a set if and only if $f(m)=n$. Similarly for 'g'. (See [15] for the idea to take Fregean senses to be algorithms.). The two expressions 'f' and 'g' might have different senses but the same reference, if the algorithms associated with them are different but extensionally equivalent, in the sense that they always return the same output for a given input. However, there is not algorithm to determine whether different algorithms are extensionally equivalent, on account of Rice's theorem, a fundamental result in computability theory [16, Corollary 1.6.14]. (For another philosophical application of Rice's theorem, see [17]).

Frege never said that the sense of a term for a numerical function is an algorithm and some of his writing suggest that he might have rejected the identification of the referent of a functional term with a set (of pairs of natural numbers). However, the proposed reinterpretation of Frege's claim still aligns with many aspects of Frege's

distinction between sense and reference. The referent of a functional term is a function, i.e., a mapping of (sequences of) natural numbers with natural numbers. An algorithm that computes a function is a way of presenting that mapping, a path, a recipe that can be followed to find the value of a function for a given input. Hence the algorithm associated with a functional term is a way of presenting the referent of that functional term. Hence, Rice's theorem can be paraphrased as a vindication of a version of Frege's claim, in this way: "Complete knowledge of the referent of a functional term, i.e., a function, would require us to be able to say whether every given algorithm computes that function. To such knowledge we never attain".

Conclusions

The connection between artificial intelligence and engineering is obvious. In this paper, we have emphasized the connection between artificial intelligence and conceptual engineering, aka philosophy. (On conceptual engineering, see [18–20].) This reminds us of an important point: philosophy can have an impact outside academia.

Casati R. [21] discusses several examples of philosophical debates where the participants were not professional philosophers and the context of the debate was non-academic. One example is the discussion of what counts as art that took place during a trial. Defining "art" was important in that context for a legal reason: artworks were exempted from certain forms of taxation, whereas other artifacts were not. Another example discussed by Casati is the debate about how to define "family" that took place during the writing of the Italian Constitution. Which conception of family a state adopts is not a purely theoretical issue: it has a huge impact on the lives of its citizens.

Our previous discussion suggests yet another example of a philosophical debate that had repercussions outside the seminar room: the way we defined the notion of solvability had a huge impact on the world we are living in.

Artificial intelligence plays an important role in contemporary medicine. Philosophy played a role in the development of artificial intelligence. The general lesson is that philosophical and scientific progress are connected. Philosophical progress is achieved when we improve our concepts, and working with better concepts leads to scientific progress. Keep this in mind the next time you encounter a philosopher of medice working on the notion of "health".

Research funding: Matteo Plebani acknowledges that the research activity that led to the realization of this paper was carried out within the Department of Excellence Project of the Department of Philosophy and Education Sciences of the University of Turin (ex L. 232/2016). Other relevant projects: Proyecto (FFI2017-82534-P) from FEDER/Ministerio de Ciencia, Innovación y Universidades-Agencia Estatal de Investigación and PID2020-115482GB-I00 from Ministerio de Ciencia e Innovación/Agencia nacional de Investigación.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Competing interest: Authors state no conflict of interest.

Informed consent: Not applicable.

Ethical approval: Not applicable.

References

1. Soare RI. The history and concept of computability. In: Griffor E, editor. Handbook of computability theory; 1999, vol 140:3–36 pp.
2. Sieg W. On computability. In: Irvine A, editor. Handbook of philosophy of science. Philosophy of mathematics. Citeseer; 2009:549–630 pp.
3. Gödel K. Remarks before the princeton bicentennial conference on problems in mathematics. In: Feferman S, Dawson J, Kleene S, editors. Kurt gödel: collected works vol. II. Oxford: Oxford University Press; 1946:150–3 pp.
4. Turing A. On computable numbers, with an application to the entscheidungsproblem. Proc Lond Math Soc 1936;42:230–65.
5. Cubitt T, Perez-Garcia D, Wolf MM. Undecidability of the spectral gap. Forum of Mathematics Pi 2022;e14:1–102.
6. Odifreddi P. Classical recursion theory: the theory of functions and sets of natural numbers. Amsterdam, North Holland: Elsevier; 1989.
7. McGee V. Why study computability? Available from: ocw.mit.edu/courses/24-242-logic-ii-spring-2004/resources/why_study_comptt.
8. Incurvati L. Conceptions of set and the foundations of mathematics. Cambridge: Cambridge University Press; 2020.
9. Shapiro S. Computability, proof, and open-texture. In: Olszewski A, Wolenski J, Janusz R, editors. Church's thesis after 70 years. Berlin, Boston: De Gruyter; 2013:420–455 pp.
10. Lucas JR. Minds, machines and gödel. Philosophy 1961;36:112–27.
11. Franzén T. Gödel's theorem: an incomplete guide to its use and abuse. Wellesley, MA: AK Peters/CRC Press; 2005.
12. McGee V. In: Fitting M, Rayman B, editors. Gödel, lucas, and the soul-searching selfie. Cham: Springer International Publishing; 2017:147–63 pp.
13. Horsten L, Welch P. Godel's disjunction: the scope and limits of mathematical knowledge. Oxford, England: Oxford University Press UK; 2016.
14. Frege G. Sense and reference. Phil Rev 1948;57:209–30.
15. Moschovakis YN. Sense and denotation as algorithm and value. In Oikkonen J, Vaananen J, editors. Lecture notes in logic. Berlin: Springer; 1994, vol 2:210–49 pp.

16. Soare RI. Turing computability. Theory and applications of computability. Berlin, Heidelberg: Springer; 2016.
17. Plebani M, San Mauro L, Venturi G. Thin objects are not transparent. Theoria 2021;1–12. https://doi.org/10.1111/theo.12373.
18. Cappelen H. Fixing language: an essay on conceptual engineering. Oxford: Oxford University Press; 2018.
19. Haslanger S. Gender and race: (what) are they? (What) do we want them to Be? Noûs 2000;34:31–55.
20. Plunkett D, Cappelen H. A guided tour of conceptual engineering and conceptual ethics. In: Cappelen H, Plunkett D, Burgess A, editors. Conceptual engineering and conceptual ethics. Oxford: Oxford University Press; 2020:1–26 pp.
21. Casati R. Prima lezione di filosofia. Roma-Bari: Laterza; 2011.