Tito Marques Palmeiro*

DOI: http://doi.org/10.32334/oqnfp.2021n49a792

What can Philosophy say, in principle, about Computers?

O que a Filosofia pode dizer, em princípio, sobre os Computadores?

Astract

Computers are the unexpected outcome of mathematical investigations from the first half of the 20th century. From mathematics to physics, and even to biology, there are many scientific disciplines in charge of their development nowadays; however, the same cannot be said about philosophy. My purpose is to understand whether philosophy would have something relevant to say about computers, even though it does not play any relevant role in this new endeavor. I consider that in order to answer this question, philosophical inquiry must discuss, in the first place, the work of Alan Turing. He created the concept of computer in a 1936-1937 paper, "On Computable Numbers, with an Application to the Entscheidungsproblem*", and he also reflected upon the extreme implications of this concept on later texts, as in a 1947 lecture on the Automatic Computing Engine (ACE), where he expressed some interesting possibilities for understanding how computers and philosophy relate to each other. I intend to show therewith that his work offers some important insights for answering the decisive philosophical question on this subject:* What can philosophy say, in principle, about computers?

Keywords: Computers; Philosophy; Alan Turing; Errors

* Universidade do Estado do Rio de Janeiro (UERJ). Contato: tito22mp@gmail.com

Recebido em: 04/04/2021 Aceito em: 10/11/2021

Resumo
Os computadores são o resultado inesperado de investigações matemáticas da primeira metade do século XX. Da matemática à física, e mesmo à biologia, há muitas disciplinas científicas encarregadas de seu desenvolvimento hoje em dia; no entanto, o mesmo não pode ser dito sobre a filosofia. Meu objetivo é entender se a filosofia teria algo relevante a dizer sobre os computadores, ainda que não tenha papel relevante nessa nova empreitada. Considero que, para responder a essa pergunta, a investigação filosófica deve discutir, em primeiro lugar, a obra de Alan Turing. Ele criou o conceito de computador em um artigo de 1936-1937, "On Computable Numbers, with an Application to the Entscheidungsproblem*", e também refletiu sobre as implicações extremas desse conceito em textos posteriores, como em uma palestra de 1947 sobre o Automatic Computing Engine (ACE), onde expressou algumas possibilidades interessantes para entender como os computadores e a filosofia se relacionam. Pretendo mostrar com isso que sua obra oferece alguns insights importantes para responder à questão filosófica decisiva sobre esse assunto:* O que a filosofia pode dizer, em princípio, sobre os computadores?

Palavras-chave: Computadores; Filosofia; Alan Turing; Erros

Computers are the unexpected outcome of mathematical investigations developed in the 1930s by mathematicians such as Alan Turing, Emil Post or Alonzo Church. Since then, science and technology have associated their efforts towards a new endeavor that is changing the world around us. As in previous technological revolutions, its full implications cannot be dealt with exclusively by the disciplines in charge of it, and therefore it seems possible to ask what something apparently foreign to this revolution, as philosophy, would have to say about it. It is true that any contribution helping to cope with the many problems aroused by the dissemination of computers in our lives is welcome, but this does not mean that philosophy should be taken for a handmaiden of science and technology, condemned to the Sisyphean role of continually analyzing their impressive developments. I intend to show that philosophy has something much more decisive to do than to follow an endless chain of consequences, for it is an inquiry directed to what is first and foremost in every problem.

In order to accomplish this task, I propose to discuss the work of Alan Turing. Turing created the concept of computer in his 1936-37 seminal paper, 'On Computable Numbers with an Application to the *Entscheidungsproblem*'.[1] From 1939 on, he not only worked on two real computing devices (the Bletchley Park Bombe and the National Physical Laboratory ACE), but he also reflected upon the theoretical implications of this concept. The following pages concentrate on some passages of his work discussing its most extreme theoretical implications. I shall start with a 1947 lecture on the first general purpose computer he designed, the Automatic Computing Engine (ACE). There he makes two intriguing and decisive claims:

> *I expect that digital computing machines will eventually stimulate a considerable interest in symbolic logic and in mathematical philosophy. (Turing 1947, p. 12)*

In order to make this passage clearer, it is necessary to discuss the meaning of the expressions 'symbolic logic' and 'mathematical philosophy'. This will be done in the next two sections (*1* and *2*). It is expected that this discussion will contribute to the understanding of the possibility of a philosophical discourse on computers, which will be explored in the subsequent sections (*3*, *4*, and *5*).

1. A claim about symbolic logic

Concerning the expression 'symbolic logic,' it seems possible to consider it as self-explanatory since there are many books on the history of logic dealing with the progressive use of symbols, from its pre-history in the works of Aristotle, Plato and Greek mathematicians to modern and contemporary achievements. However, Turing employed the expression 'symbolic logic' in a different context — that of a lecture on a real computing device — and with an entirely different meaning:

> *The language in which one communicates with these machines, i. e.* the language of instruction tables, *forms* a sort of symbolic logic. *The machine interprets whatever it is told in a quite definite manner without any*

1 The machine Turing proposed was soon called by Alonzo Church a 'Turing Machine' (Church 1937, p. 43).

> *sense of humour or sense of proportion. [...] [I]n principle one should be able to communicate in* any symbolic logic, *provided that the machine were given instruction tables which would enable it to interpret that logical system. (Turing 1947, p. 12, emphasis mine)*

This expression has both a negative and a positive sense in this passage. On the one hand, Turing clearly denies that 'symbolic logic' would be limited to a specific set of symbols or a given syntax, but, on the other hand, he positively identifies it to 'the language of instruction tables'. We may conclude therefore that this expression means: any possible language allowing the achievement of tasks by computers, i.e., 'symbolic logic' stands for the *general idea of programming languages*. It is, therefore, necessary to ask why Turing claimed that computers would probably stimulate interest in programming languages, and why he did not make a similar claim about hardware. This would be quite understandable since this lecture is mainly devoted to the physical features of the *ACE*, such as memory storage, input, output or the arithmetic unit circuit. However, while hardware problems can be solved by the use of engineering techniques, there is no general technique available for creating programs. We do not see it clearly nowadays because programmers are taught best practices for solving stereotyped problems like sorting, looping and so on. Due to his previous programming experience, Turing was aware that to write a program is quite different from applying our knowledge of physics to create artifacts like 'bridge,' 'memory storage' or 'arithmetic unit circuit'. He stated this special difficulty three years later in a programmer's handbook for a new computer model in which he worked:

> *Programming is a skill best acquired by practice and example rather than from books. The remarks given here are therefore quite inadequate. (Turing 1951, p. 51)*

After fifteen years of programming experience, Turing considered important to state as clearly as possible the impossibility of teaching how to write programs. This stands in sharp opposition to the idea — implicitly at stake nowadays — sustaining the plethoric offer of courses and books on this subject. The negative statement of Turing means that among the many questions to be solved for computer development, the most defying one is related to the creation of programs as a means for allowing them do whatever we may wish for — and this can be possibly the reason beneath the first claim on programming languages. The main scientific and technological question guiding the development of computers can be stated as: '*How can we make computers do that?*', which is mostly answered by the creation of programs to solve specific problems, like effecting visual recognition, or computing huge amounts of data. In the sequel, I shall compare this question to an entirely different one, associated with the second expression employed in this lecture: 'mathematical philosophy'.

2. On mathematical philosophy

To understand the meaning of the expression 'mathematical philosophy,' I will follow the thread offered by a description of the theoretical landscape around Turing:

> *The main schools of mathematical philosophy at the beginning of this century were Russell and Whitehead's view that logic was the basis for everything, the formalist school of Hilbert, and an 'intuitionist' constructivist school of Brouwer. (Chaitin 2007, p. 96)*

This passage reduces the possibilities of understanding 'mathematical philosophy' to three schools. We could maybe guess that Turing was possibly referring to the first one, in particular to Russell's 1919 popularization book *Introduction to mathematical philosophy.* This may be a good bet as, according to Charles Petzold, he actually read it in 1933.[2] Russell said in this book some interesting things that may help us understand the general question at stake here, about the possibility of a philosophical discourse on computers. He stated in the introduction that 'the matters concerned' in mathematical philosophy 'were included in philosophy so long as no satisfactory science of them existed,' and he gave as examples 'the nature of infinity and continuity' (Russell 1993, p. v). Philosophy, however, which since the Greeks had the role of defining mathematical principles, does not have in fact for Russell the required expertise to fully understand them, and for this he proposed *Mathematical philosophy* as a transformation of the traditional philosophy of mathematics. This new discipline would speculate from an acquaintance of 'the more scientific parts of the principles of mathematics'. However, it is still different from mathematics itself because '*Mathematical philosophy*, in the strict sense, cannot, perhaps, be

2 Petzold 2008, p. 60: 'Turing's interest in the rarefied world of mathematical logic might have begun in 1933 when he read Bertrand Russell's 1919 work *Introduction to Mathematical Philosophy*'.

held to include such definite scientific results as have been obtained in [mathematics]' (Russell 1993, p. v). It does not have the same degree of certainty of mathematics because it must inquire into that which mathematics cannot:

> *philosophy of mathematics will naturally be expected to deal with questions on the frontier of knowledge, as to which comparative certainty is not yet attained. (Russell 1993, p. v)*

However, we must not rely entirely on Russell's discussion of 'mathematical philosophy' because it is much more important to go back to Turing's lecture on the *ACE* to ask what he may have said about those 'questions on the frontier of knowledge'. That is how he explained this expression:

> *As regards mathematical philosophy, since the machines will be doing more and more mathematics themselves, the centre of gravity of the human interest will be driven further and further into philosophical questions of what can in principle be done etc. (Turing 1947, p. 12)*

Unlike Russell, Turing did not take 'mathematical philosophy' for a new version of the traditional philosophy of mathematics that would better ground mathematical principles. He takes it rather literally: as an association of 'machines doing mathematics' to 'philosophy'. The probable 'interest in [...] mathematical philosophy' is therefore rooted in the philosophical questions that may rise as the side-effect of 'machines [...] doing more and more mathematics'. Turing seems to indicate thereby something quite unexpected: that the widespread use of computers in the most diverse fields of activity would ask, by itself, for philosophy.

3. 'How can we make computers do that?'

How is it possible to take seriously a claim stating that the development of computers would create space for philosophy? It is hard to accept this even as a mere theoretical hypothesis since philosophy has never played any substantial role on the long history of computer industry. The apparent strangeness of this claim is rooted in the fact that the question '*How can we make computers do that?*' has nothing to do with philosophy, as it is continually answered by science and put into practice by the technological means available at a given moment. Turing was well aware of this fact because his answer, offered in the 1945 proposition for the creation of the *ACE* digital computer, made no reference to philosophy at all:

> *The class of problems capable of solution by the machine can be defined fairly specifically. They are those problems which can be solved by human clerical labour, working to fixed rules, and without understanding. (Turing 1945, p. 14)*

Just as any human being acting 'without understanding,' computers work with 'fixed rules'. We can make them do anything that can be achieved by us in a series of mechanical steps. However, this answer does not stand for something new in Turing's work as it is only the reassessment of one important achievement of his seminal 1936-37 paper in which he showed that the question '*How can we make computers do that?*' is answered by the creation of instruction tables, i.e., by programming the computer. We will see in the sequel that the second claim expressed in this lecture has an entirely different scope. To see it clearly, let us quote again the last passage from the lecture on the *ACE* and stress the decisive words:

> *As regards mathematical philosophy, since the machines will be doing more and more mathematics themselves, the centre of gravity of the human interest will be driven further and further into* philosophical questions *of what can* in principle *be done etc. (Turing 1947, p. 12, emphasis mine)*

While Turing's first claim is directed to the scientific question responsible for computer development, the second one unexpectedly introduces the possibility of thinking philosophically about them. The many questions asking '*What can computers do in principle?*' are called 'philosophical' because they cannot be answered by the same means available for answering the first one, that is, by 'doing more and more mathematics'. Turing was certainly right in supposing that the widespread use of computers would stimulate an interest in programming languages and in theoretical and technological problems related to them, but he would be probably very upset that almost a century after the creation of the concept of computer and of the creation of real computing devices, 'the centre of gravity of the human interest' is still not driven into philosophical questions. As a consequence, the main discourses on computers tend to concentrate on '*How can we make computers do that?*' and we think of them accordingly in terms of efficiency, connectivity, memory management, and so on. However, with that positivistic approach, we do not even wonder that computers may also require an entirely different sort of approach.

4. Computers, philosophy, and errors

The previous moments have shown that the possible place for a philosophical inquiry on computers is not the same as that of the disciplines in charge of their development. Philosophy must think from a different standpoint: that one indicated by the expression 'in principle'. This does not mean that it should determine computers by a reference to a general and abstract principle, a traditional αρχὴ from which it would think computers *a priori*. Turing says nothing of that sort, for he simply indicates the existence of questions intrinsically raised by computer activity. The expression 'in principle' indicates therefore that a philosophical inquiry of computers should concentrate on the 'questions on the frontier of knowledge'. These questions have a twofold nature, for they are *1. intimately related to computer activity* , but they *2. shall never have a definitive answer.* The difficulty to envisage those questions lies in the fact that it is not possible to propose a general procedure for determining them, for they are raised by computer's very activity.

In the sequel, I propose a first philosophical question for discussion. It was implicitly raised by Turing's paper when he created the concept of computer. In his paper, he wrote many 'programs' and 'subroutines' to explain the possibility of a machine that would do the same job of a human being working with a pencil and an eraser on a sheet of paper.[3] However, as it was soon noticed, many of these 'programs' and 'subroutines' contained errors. Emil Post produced some corrections to them in the appendix of 'Recursive Unsolvability of a Problem of Thue' (Post 1947), and Donald Davies did the same in his essay 'Corrections to Turing's Universal Computing Machine' (Copeland 2004). In 1975, Davies gave an important testimony about the reaction of Turing to his discoveries. He said that he 'found a number of quite bad programming errors [...] in the specification of the machine'. He showed them to Turing expecting that he would reply with something like 'Oh fine, I'll send along an addendum' to his paper. However, Turing reacted very differently:

> *But in fact he was very annoyed, and pointed out furiously that really it didn't matter, the thing was right* in principle*, and altogether I found him extremely touchy on this subject (Copeland 2004, pp. 92-93, emphasis mine)*[4]

3 Called respectively 'tables' and 'skeleton tables'.

4 If we are to believe in the testimony of Davies that Turing would have said that the concept of computer was right 'in principle,' then we should ask accordingly if this concept is really a scien-

One may read this passage as a testimony of a personal reaction to a criticism, but it has in fact a deeper meaning. To fully understand it, we must notice that Turing never published a paper modifying his concept of computer and that he continued to work from it in his theoretical texts and in his engineering projects.[5] This means that he really considered that any possible errors in the programs he wrote would not deny the validity of the concept. However, he may have become 'extremely touchy' because the concept requires the general idea of programming languages as a means for its activity, but no one can guarantee beforehand the correction of a specific program written in a given syntax — and in fact, many of them were wrong, mostly because Turing did not have the possibility of 'compiling' and 'running' them.

As is well known, Turing proved in his 1936 paper that it is not possible to envisage a machine that would verify if others machines work properly. As a consequence, program errors stand for the kind of philosophical questions we are searching for. At a first sight, it may seem surprising to take them as standing for philosophical questions, but it must be noticed that they do respect the two requirements indicated above. In the first place, they are evidently 'intimately related to computers' (as indicated in 1.) because it is tautological to say that computer errors are *produced by computers*. This is so much the case that it seems surprising to consider that (according to 2.) they cannot have a definite answer. This seems to be simply wrong because programming languages have instructions that allow programmers to deal with errors, one of the simplest being the 'else' option in an 'if' statement, or the 'exception' directive in Java and other languages. It is true that it is possible to minimize errors by using test scenarios and statistical analysis, and it is also true that it is possible to automatize error-checking with automatic procedures designed to analyze a finite set of scenarios. However, this is only a practical and limited solution because, as Turing stated in his lecture on the *ACE*: '[T]he statistical method can only help the analyst, not replace him.' (Turing 1947, p. 20).

Let us try to understand a little bit more why programs could never really catch all possible errors. A program is defined by *what is coded* in its

tific or rather a philosophical one. However, I will not discuss this problem here.

5 It is well known that Turing wrote in 1937 a correction to his 'proof of the insolubility of the Entscheidungsproblem,' which 'contained some formal errors' (TURING 1937, p. 344), but this has no incidence on the concept of computer proposed.

instructions,[6] such as to add numbers, digitalize images, play music and so on. However, it is not possible to guarantee that it will fully accomplish what *we expect* it to do in every possible situation because the numbers typed may produce overflow, the original document to be scanned may be too faint, or the music file may require too much memory to be read. That is why good programming practices and methods of software development contain a test phase. It is true that in this phase the most important errors should be identified and eliminated, but no one can be entirely sure that this will be really the case because all epistemic relevant issues to program creation cannot be known beforehand. The fundamental point is not that programming is an activity depending on social criteria,[7] neither that the idea of formal program verification should be blamed for trying to make an *a priori* proof of an *a posterity* real machine environment.[8] The real philosophical issue is that the set of possible program states is *finite*, while that of all possible errors is *infinite*.

It is not a simple a question of a set being greater than the other, that which would possibly have a definitive answer by the use of mathematical means. What is philosophically important here is that errors are not related to the definition of computers because they stand to *what computers cannot do*. Errors mean non-solvability; they stand for *an external limit to what can be done, in principle, by computers.*

6 Davis 1972, "The program itself is the only description of what the program will do".

7 De Millo, Lipton & Perlis 1979 provoked intense reactions from the computer science community for stressing the impossibility of strict proofs of formal verifiability and for proposing accordingly that they should be replaced by social analysis of what constitutes an acceptable software: "The concept of verifiable software has been with us too long to be easily displaced. […] Verifiability is not and cannot be a dominating concern in software design. Economics, deadlines, cost-benefit ratios, personal and group style, the limits of acceptable error — all these carry immensely much more weight in design than verifiability or nonverifiability." (p. 279). For an analysis of the debate, see MacKenzie 2004, chapter 6, "Mechanizing Proof: Computing, Risk, and Trust".

8 Fetzer 1988 directed the criticism of De Millo, Lipton & Perlis 1979 to what he considered as the most radical problem at issue, that of searching for *a priori* formal verifiability proofs of an *a posteriori* environment, suitable only to inductive reasoning on empirical facts: "In this case, the absolute verification of an abstract machine is logically impossible because its intended interpretation is a target machine whose behavior might not be described by those axioms, whose truth can only be established by induction." (p. 1059).

5. A brief conclusion on computers' errors

I would like to close this inquiry by making a few remarks on the philosophical statement just proposed, that errors constitute 'an external limit to what can be done, in principle, by computers'. I intend to show that this first attempt at thinking computers philosophically indicates that philosophy has some new and interesting things to say about them. Here I shall briefly discuss one single feature introduced by program errors.

In the lack of a philosophical inquiry on computers, the presupposition beneath the question '*How can we make computers to do that?*' is a positivistic one, implying that their activity has potentially no limit. In this perspective, errors are extremely productive because they are the reason why we tend to think about computers in contradictory terms: as exact machines that can nonetheless infinitely progress. Computers are adopted in the most diverse fields, ranging from Banking and Manufacturing to the Academia, because they are considered a sound basis for the management and the production of knowledge. Errors are nonetheless bound to them as is clearly shown by the proliferation of software beta-versions, correcting patches and new program versions. What is surprising in this situation is that errors are not taken for a symptom of a possible lack of grounds of computing, but appear moreover as an important opportunity for the development of new possibilities.

In contrast to the positivistic approach frequently associated to the understanding of computing as a bold and limitless "new technology", our inquiry has shown that errors are not simple details that could be ignored due to the promise of an infinite progress in computing, but they constitute an intrinsic limit for *what computers can do*. This statement is not something that could be understood by experts only, for it is a philosophical statement directed to anyone trying to think computers thoroughly. This limit concerns the very definition of computers, as they can only be said to do what they are *supposed to do* so long as no error has been detected. Philosophy shows therefore that computers are not truth-machines, but an intriguing concept that is defined by what it *does*, and intrinsically incapable of dealing with what it *does* not.

References

CHAITIN, Gregory. *Thinking about Gödel and Turing. Essays on Complexity, 1970-2007.* Singapore: World Scientific Publishing Co., 2007.

COPELAND, Jack. *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life plus The Secrets of Enigma,* Oxford: Clarendon Press (Oxford University Press), 2004.

CHURCH, Alonzo. "Review: A. M. Turing, On Computable Numbers, with an Application to the *Entscheidungsproblem*". *Journal of Symbolic Logic* (2): 42–43, 1937.

DAVIS, P. J. Fidelity in mathematical discourse: Is one and one really two? *The American Mathematical Monthly* 79, 3 (1972), 252-263.

DE MILLO, Richard; LIPTON, Richard; PERLIS, Alan. Social Processes and Proofs of Theorems and Programs. *Communications of the ACM*, May 1979, volume 22, number 5.

FETZER, James. Program verification: The very idea. *Communications of the ACM*. September 1988, Volume 3, Number 9.

GÖDEL, Kurt. Remarks before the Princeton bicentennial conference on problems in mathematics (1946), *Collected Works, volume II, publications 1938-1974.* New York: Oxford, 1990.

MACKENZIE, Donald. *Mechanizing Proof: Computing, Risk, and Trust.* Cambridge: MIT Press, 2004.

PETZOLD, Charles. *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine.* Indianapolis: Wiley Publishing, 2008.

POST, Emil. "Recursive Unsolvability of a Problem of Thue". *The Journal of Symbolic Logic,* Vol. 12, No. 1. (Mar., 1947), pp. 1-11.

TURING, Alan. "Intelligent Machinery" (1948) in *The Essential Turing*

TURING, Alan. "Alan Turing's Manual for the Ferranti Mk. I" (1951). *http://curation.cs.manchester.ac.uk/computer50/www.computer50.org/kgill/mark1/RobertTau/turing.pdf*

TURING, Alan. Intelligent Machinery, a Heretical Theory" (1951) in *The Essential Turing*

TURING, Alan. "Lecture to the London Mathematical Society, February, 20, 1947, on ACE". AMT/B/1: *http://www.turingarchive.org/browse.php/B/1.*

TURING, Alan. *Proposed Electronic Calculator (ACE)*. (1945) AMT/C/32: *http://www.turingarchive.org/browse.php/C/32.*

RUSSELL, Bertrand. *Introduction to Mathematical Philosophy*, second edition. George Allen & Unwin, 1920; Dover, 1993.