© Lo Sguardo - rivista di filosofia
N. 34, 2022 (I) - Algoritmo

DOI: 10.5281/zenodo.7768560

Contributi/1

*Alan Turing and the Cognitive Foundation of the Concept of Algorithm**

Simone Pinna 0000-0001-9049-5585

Marco Giunti 0000-0003-4182-4850

Articolo sottoposto a doppia blind peer review. Inviato il 08/12/2021. Accettato il 08/06/2022.

The work of Alan Turing (1936) set a milestone for the foundation of the concept of algorithm by grounding the notion of effective procedure on a special type of real cognitive phenomenon, namely, that of a human being performing rule-based symbolic transformations with the only aid of paper and pencil. In this work, after a brief historical overview, we show how Turing arrived at a negative solution of the decidability problem for first order logic and in which sense Turing's explication of the intuitive concept of effective procedure is sufficient to justify Church's Thesis. We then present a cognitive interpretation of Turing's theory of computation, according to which Turing machines are viewed as models of real phenomena of mind-environment interaction.

***

Introduction

During the first decades of the 20th century a vast group of mathematicians and logicians was involved in the precise definition of a series of foundational concepts for mathematics. The theoretical background of these studies was the so-called logicist project, i.e., the attempt to reduce all mathematics to logic.[1] Logicists notably recognized Cantor's set theory as a starting point for a rigorous redefinition of all mathematical theories in logical terms.

Bertrand Russell's discovery of a contradiction in Cantor's set theory (the Russell's antinomy) marked the failure of the original logicist project. However, the studies on the foundation of mathematics gained renewed impetus at the

* Supported by Fondazione Banco di Sardegna (FdS 2019, research grant n. F72F20000420007).
[1] G. Frege, *The Foundations of Arithmetic: A Logico-mathematical Enquiry into the Concept of Number*, Northwestern University Press 1968.

beginning of the 20th century thanks to David Hilbert's work. According to Hilbert's formalist project, any mathematical theory should be reduced to a pure symbolic system, such that one must be able, only through the means provided by the system, to prove the consistency of the theory. An outstanding problem in the formalist framework is the problem of decidability (*Entscheidungsproblem*), proposed in its most general version by Hilbert and Hackermann[2]: Is there any effective procedure (or any algorithm) for deciding whether an arbitrary first-order language statement is universally valid (i.e., a tautology, or a logical truth) or not?

An important side problem of the Entscheidungsproblem is the rigorous definition of the intuitive notion of effective calculability, or to find an adequate method to set apart computable numerical functions from non-computable ones, where computable numerical functions are those for which there exists an algorithm (or an effective procedure) that, for any argument, returns the value of the function in a finite number of steps. (For example, the function that returns the *n*-th decimal digit of π is computable, but for most real numbers the analogous function is not.)

In 1936,[3] Alonzo Church proposed the λ-calculus as a way to express in a rigorous formal manner the intuitive concept of effectively calculable function. He proved that the class of functions defined within his calculus is in fact equivalent to the class of recursive functions defined by Gödel.[4] He then proposed the following thesis (Church's Thesis): «We now define the notion […] of an effectively calculable function of positive integers by identifying it with the notion of a recursive function of positive integers (or of a λ-definable function of positive integers)».

According to Church, the fact that either recursiveness or λ-definability captures the notion of effective calculability rests on intuitive considerations. However, it is very difficult to find any similarity between a logical system such as the λ-calculus and the ways human beings execute algorithms.

The work of Alan Turing (1936)[5] set a milestone on this debate by grounding the intuitive notion of algorithm on a special type of real cognitive phenomenon, namely, that of a human being performing rule-based symbolic transformations with the only aid of paper and pencil. In this work, after a brief historical overview, we show how Turing arrived at a negative solution of the Entscheidungsproblem, and in which sense Turing's explication of the intuitive concept of algorithm is sufficient to justify Church's Thesis. We then propose a cognitive interpretation of Turing's theory of computation, according to which

[2] D. Hilbert, W. Ackermann , *Grundzüge der theoretischen logik*, Springer Verlag, Berlin 1928.
[3] A. Church, *An unsolvable problem of elementary number theory*, «Journal of Mathematics», 58, 1936, pp. 345-363.
[4] K. Gödel, *On Undecidable Propositions of Formal Mathematical Systems*, 1934, in Davis, M. (ed.), The Undecidable, Raven Press, Hewlett, New York, 1965, pp. 39-74.
[5] A. M. Turing, *On computable numbers, with an application to the* Entscheidungsproblem, «Proceedings of the London Mathematical Society» (pp. 230–265), London 1936, Oxford Journals.

Turing machines are viewed as models of real phenomena of mind-environment interaction.[6]

1. In Search of a Formal Notion of Algorithm

The most general formulation of the problem of decidability is the following: Is there any *effective procedure* to decide, for an arbitrary first-order language statement, whether it is *universally valid* (i.e., a tautology), or not? According to Hilbert, this was the most important problem of mathematical logic. Not only was first order logic at issue, but also any axiomatic theory based on a finite number of axioms. For this kind of theories, indeed, the solution of the problem of decidability would immediately yield an effective procedure to determine whether any statement of the theory is a logical consequence of the axioms or not.

In order to tackle the *Entscheidungsproblem*, however, a preliminary problem to be solved was the formulation of a satisfying formal definition of the intuitive concept of an *effective procedure*, or an algorithm. Intuitively, an algorithm (or a mechanical or effective procedure) is a finite set of clear-cut formal instructions for symbol manipulation. Given a finite collection of initial data, a human being must be able to carry out such instructions in a definite sequence of steps, with the exclusive aid of paper and pencil (or similar external supports or devices), and without resorting to any special insight or ingenuity. Note that, if a formal definition of algorithm were available, it would then be possible to formally define the computable functions, namely, those functions for which effective procedures to calculate their values do exist.

The solution to this preliminary problem is also clearly connected to the problem of decidability, for the latter presupposes a clear definition of the notion of an effective procedure. Moreover, once we have such a formal definition at hand, we will be able to try out the following strategy: if an effective procedure to decide the universal validity of any first order language statement *does not exist*, then we may be able to actually produce a first order language statement for which *we can prove* that there is no algorithm to decide whether it is universally valid or not (i.e., whether it is a tautology or not).

1.1 Church's Thesis

In 1936 Alonzo Church proposed λ-calculus as a formal method to rigorously express the intuitive notion of an effectively calculable function.[7] First, he proved that the class of λ-definable functions (namely, the functions

[6] M. Giunti, S. Pinna, *Toward a dynamical theory of human computation*, «Logic Journal of the IGPL», 24 (4), 2016, pp. 557-569; Wells, A., *Rethinking cognitive computation: Turing and the science of the mind* (hereafter, Wells 2005), Basingstoke, UK (2005), Palgrave Macmillan.
[7] A. Church, *An unsolvable problem of elementary number theory*, cit.

definable in his λ-calculus) is equivalent to the class of *recursive* functions, previously defined by Kurt Gödel). Then, he proposed the following thesis (known thenceforth as *Church's Thesis*):

> We now define the notion, already discussed, of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers (or of a λ-definable function of positive integers).
>
> This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion.[8]

Church argued that an algorithm for computing a function exists if and only if that function can be expressed as a string of symbols written in his logical language or, equivalently, as a recursive function. However, he recognized that he could not prove this claim, for there is no secure way (i.e., a formal proof) to justify the identification of a formal definition with an intuitive notion. In other words, this justification can only be obtained by an intuitive stance.

Here, we cannot give an even cursory presentation of Church's λ-calculus. However, some brief remarks on the equivalent theory of the recursive functions may be useful to better understand the difference between Church's and Gödel's approach on the one hand, and Turing's one on the other. Similar to Church's λ-definability, recursivity theory provides us with a formal system to define a class of computable functions. First of all, we start from the following three basic *primitive recursive* functions:

[1] $z(n) = 0$ (*zero* function);
[2] $id(n) = n$ (*identity* function);
[3] $s(n) = n+1$ (*successor* function).

Then, all other primitive recursive functions are defined by the following rules:

Composition: a function is primitive recursive if it can be defined by the application of a primitive recursive function to others primitive recursive functions.

Recursion: a function $f(x_1, \ldots, x_n)$ of $n \geq 1$ arguments is primitive recursive if the following two conditions are met:

[i] $f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n)$, where, if f has at least two arguments, g is a primitive recursive function and some of its arguments may be missing. If f is a single argument function, g is a zero argument function, namely, it is a natural number;

[8] *Ibid.*, p. 356.

[ii] $f(s(x_1), x_2, \ldots, x_n) = h(f(x_1, x_2, \ldots, x_n), x_1, x_2, \ldots, x_n)$, where h is a primitive recursive function and some of its arguments, except the first, may be missing.

Closure: nothing else is a primitive recursive function.

In a recursive definition the value of a function is the result of the application of the same function to smaller arguments, in a process named *recursion*. Consider the definition of the two arguments function *addition*, denoted by $a(x_1, x_2)$, as a primitive recursive one:

[4] $a(0, x_2) = id(x_2)$
[5] $a(s(x_1), x_2) = s(a(x_1, x_2))$

The function $a(x_1, x_2)$ is primitive recursive, for it meets both conditions of the rule of recursion. Indeed, condition [i] is met by equation [4], for *id* is primitive recursive; moreover, condition [ii] is met by equation [5] for, in the second member of the equation, s is a primitive recursive function with a single argument.

We now illustrate by an example how the recursive definition of addition is used in calculation. Let us take $x_1 = 2$ and $x_2 = 2$. By equation [3] we know that $s(2) = 3$, then, with this setting (by equation [5]), we are going to calculate the sum 3+2.

We start the computation by applying equation [5] to the chosen values of x_1 and x_2, then we decrease x_1 by 1 at each step, until $x_1 = 0$, as in the following series of equations:

$a(3, 2) = s(a(2, 2))$
$a(2, 2) = s(a(1, 2))$
$a(1, 2) = s(a(0, 2))$

Here, by [4] and [2], we arrive at the only result explicitly defined by the function, namely, the case where $x_1 = 0$. Knowing that $a(0, x_2) = x_2$, we can now substitute the result thus obtained for the case $x_1 = 0$; then, we increase x_1 by 1 at each step, until $x_1 = 2$, as in the following series of equations:

$a(1, 2) = s(2) = 3$
$a(2, 2) = s(3) = 4$
$a(3, 2) = s(4) = 5$

Here we have focused only on primitive recursive functions, but to cover all recursive functions we would have to present the more complex theory of *general* recursive functions, which is beyond the scope of the present paper. However, the previous explanation is sufficient for our purpose, that is, to provide the reader with a clear understanding of the peculiar kind of procedure by which a recursive function is computed.

We have seen above that a justification of Church's Thesis can be obtained only from an intuitive point of view. However, the fact that, *for any function computable by an effective procedure*, this function must also be definable in terms of recursiveness (or, equivalently, of λ-definability) is not intuitively implied by the notion of a recursive (or λ-definable) function. Indeed, the formal methods individuated by Church and Gödel are highly abstract and profoundly different from the calculation procedures we normally use (for example, the algorithms used for carrying out simple arithmetical operations with paper and pencil). Due to these reasons, Church's Thesis was initially deemed as *materially inadequate* and, ultimately, *not adequately justified*, because it lacked a sufficiently strong argument to ensure that all the numerical functions computable by a human being through the application of appropriate rules were captured by the class of recursive or λ-definable functions.

1.2 Turing's analysis of human computation

Turing's work[9] tackled the problem of explicating the notion of effective calculability by a totally different stance. His major concern was the definition of a theory of computation rigorously modeled on concrete phenomena of human computation, namely, on the actual procedures carried out by a human being that is calculating with the aid of paper and pencil. The analysis of this kind of phenomena needs a previous consideration of human cognitive abilities and limits.

Turing argues that his computing machines (known thereafter as *Turing machines*) capture the intuitive notion of an algorithm. This claim is justified by an *analogical argument*, namely, an inductive argument which correlates two or more phenomena in order to substantiate the hypothesis that they share some characteristic property. For example, (1) consider an object *a* whose characteristic property *p* we are interested in is known, then (2) notice that some essential features of *a* are similar to those of another object *b*; given these similarities we can claim that (3) *a* and *b* likely share *p*.

Turing's argument starts from the consideration of a *type* of real cognitive phenomenon, namely, the fundamental operations that we can observe in a human being carrying out a computation with paper and pencil. Turing explicitly refers to this kind of phenomena, as in the following quotation:

> Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow

[9] A. M. Turing, *On computable numbers*, cit.

an infinity of symbols, then there would be symbols differing to an arbitrarily small extent.[10]

In this passage, Turing focuses on the individuation of the fundamental elements of human computation, by eliminating all those features that are *inessential* to the calculations performed by a human being. This is to be intended in the sense that they *complicate the algorithm* without yielding any gain with respect to the *results* that can be computed. The two-dimensional paper sheet is thus considered inessential in this sense, for it is always replaceable by a unidimensional tape divided into squares. By contrast, the assumption that the number of symbols that a 'computer' (i.e., according to Turing's terminology, a *human being* who computes) is able to recognize at a glance must be finite is essential, for it directly depends on human cognitive limits. Another inessential feature is the compositional character of the mental states used while computing.

> We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. […] Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.[11]

It is hard to avoid the impression that Turing is considering the physical limitations we need to impose on a computing *machine* rather than the cognitive limitations of a computing *human being*. This is partly due to our present usage of the word 'computer', as a synonym of 'digital computer'. However, the concept of a digital computer as the physical machine we are used to was unknown in the mid-1930s. A thorough reading of Turing's argumentation should convince even the most skeptical reader. Consider, for example, the following passage on the identifiability of symbols:

> In most mathematical papers the equations and theorems are numbered. Normally the numbers do not go beyond (say) 1000. It is, therefore, possible to recognize a theorem at a glance by its number. But if the paper was very long, we might reach Theorem 157767733443477; then, further on in the paper, we might find "... hence (applying Theorem 157767733443477) we have […]". In order to make sure which was the relevant theorem we should have to compare the two numbers figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice.[12]

It is obvious that this argument makes sense only if we consider human cognitive limits rather than general physical limits, we can impose on a computing machine. After the identification of the essential elements of computation,

[10] *Ibid.,* pp. 249.
[11] *Ibid.,* p. 250.
[12] *Ibid.,* p. 251.

Turing analyzes the behavior of a computer that conforms to all the imposed limitations.

> The behaviour of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment. [...]
>
> Let us imagine the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer. We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes of this kind.[13]

Now Turing describes how to build into a computational model all the elements needed to describe the essential work of a human computer, namely, the simple operations drawn from the analysis of the real cognitive phenomenon he considers.

> The simple operations must therefore include:
>
> (a) Changes of the symbol on one of the observed squares.
>
> (b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.
>
> It may be that some of these changes necessarily involve a change of state of mind. The most general single operation must therefore be taken to be one of the following:
>
> (A) A possible change (a) of symbol together with a possible change of state of mind.
>
> (B) A possible change (b) of observed squares, together with a possible change of state of mind.
>
> The operation actually performed is determined [...] by the state of mind of the computer and the observed symbols. In particular, they determine the state of mind of the computer after the operation is carried out.
>
> We may now construct a machine to do the work of this computer.[14]

All the functional features of Turing's computational models listed in this quotation are drawn from his analysis of human computation. The intuitive force of his definition of an effective procedure, in fact, arises from the analogy between the description of a class of real phenomena and the operations carried out by his ideal models. The (negative) solution to the problem of decidability

[13] *Ibid.,* p. 250.
[14] *Ibid.,* p. 251.

he proposes (see §1.3) is based on the same analogy. Indeed, by this analogical argument it is possible to justify the general thesis according to which Turing machines can execute all and only the essential computational procedures that a human being can, in principle, carry out in a finite amount of time with the only aid of paper and pencil or similar external resources, and without resorting to any special insight or ingenuity. In a nutshell, *Turing's Thesis* can be formulated as follows: *Turing machines are adequate ideal models of human computation.*

It is worth noting that the class of computable functions individuated by Turing is identical to that individuated by Church. Turing himself[15] proved that the class of all numerical functions computable by his machines (henceforth, *Turing-computable* functions) is identical to the class of λ-definable ones. Turing's argument, then, also supplies a strong intuitive justification to Church's Thesis, overcoming the theoretical impasse described above (§1.1).

2. Turing's Negative Solution of the *Entscheidungsproblem*

Turing's strategy for the negative solution of the decidability problem consists of two main moves. First, he identifies the intuitive notion of an effective procedure with the formal operations carried out by his computing machines (Turing's Thesis), and intuitively justifies this identification through an analogical argument (§1.2). Second, he employs this identification to justify Church's Thesis, which can be stated as the claim that the class of all computable functions (in the intuitive sense) is a subclass of the class of all Turing-computable functions. This finally allows him to prove by *reductio* that there is *no algorithm* that decides whether an arbitrary first order sentence is a logical truth (or a tautology). Turing remarks[16] that his result is sensibly different from the incompleteness result obtained by Gödel.[17] While the latter proved that any consistent minimally expressive axiomatic theory[18] is incomplete – for there is always a particular sentence of the theory that cannot be proved or disproved within the theory itself – Turing shows that there is no *algorithm* capable to decide, for any sentence of a first order language, whether it is universally valid.

2.1 Essential elements of Turing machines

To fully understand Turing's proof, it is necessary to briefly introduce the notion of a Turing machine (*TM*). A *TM* is made up of the following components:

[15] *Ibid., Appendix*.
[16] *Ibid.,* p. 259.
[17] K. Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, «Monatshefte für Mathematik und Physik», 38 (1931), 173-198.
[18] For a 'minimally expressive axiomatic theory' we mean a theory that is able to express and prove at least what can be proved in elementary arithmetic (see M. L. Dalla Chiara, *La logica*, Milano 1974, §1.5).

1. A finite automaton consisting of

- a simple input-output device that implements a specific set of instructions (machine table);

- an internal memory that holds only one discrete element at each step (internal state) and

- an internal device (read/write head) that can scan and change the content of the internal memory.

2. An external memory consisting of a tape divided into squares, potentially extendible in both directions *ad libitum*;

3. an external device (read/write/move head) that scans the content of a cell at a time and allows the finite automaton to work on the memory tape.

At each step of a computation, the external device scans one cell of the tape and changes its content in accordance with the symbol written on the cell and its present internal state (as scanned by the read/write head in the internal memory). The rules for symbol transformations are specified in the machine table, which is a non-empty set of *quintuples* of the form $<q_i, s_j : s_k, M, q_r>$. No two quintuples of the machine table can begin with the same pair of symbols $<q_i, s_j>$. The pair $<q_i, s_j>$ in the first part of a quintuple represents the input, where the first symbol is taken from a finite set $Q = \{ q_1, q_2, \ldots, q_n \}$ of internal states and the second from another finite set $S = \{ s_1, s_2, \ldots, s_m, b \}$ of symbols, which represents the vocabulary of the tape and includes at least two symbols, one of which, i.e., b (the *blank*), means that the content of the cell is null. The number of tape cells is always finite, but it can be increased as needed by adding blank cells. The triple $< s_k, M, q_r >$ in the second part of a quintuple represents the output, where the first and the third symbol are taken, respectively, from the above mentioned sets $\{ S \}$ and $\{ Q \}$, while M represents one of the possible movements of the external head on an adjacent cell of the tape with respect to the presently scanned one (L for 'move left' and for 'move right') or a null movement (H for 'halt'). Each single quintuple of a machine table, then, can be interpreted as a conditional instruction that tells the machine: 'if you read q_i in your internal memory and s_j on the tape, then substitute s_k for s_j on the tape, q_r for q_i in the internal memory, and move the external head according to M'. The machine stops computing if and only if the current internal state q_i and the scanned symbol s_j do not correspond to the initial pair of any quintuple in the machine table, or the corresponding quintuple has the form $< q_i, s_j : s_j, H, q_i >$.

Turing introduces some important conventions in his notation. He distinguishes between two types of symbols that form the vocabulary of the tape: 'symbols of the first type', or *figures*, are only 0 and 1 – they are used to encode the binary representation of an arbitrary real number. All the others are 'symbols of the second type'. Let an arbitrary *TM* start computing from a given initial state on a blank tape. The sequence of all figures that the machine writes on the tape while it is computing is called the *sequence computed by the machine*. Any

such sequence can be interpreted as the binary representation of a real number,[19] which can thus be said to be computed by the machine as well. A *TM* is called *circular* if, when started on a blank tape, it eventually stops computing or it writes at most a finite number of figures. Otherwise, it is said to be *circle free*. An infinite sequence of figures (or the corresponding real number) is *computable* if there is a circle free *TM* such that, when started on a completely blank tape, produces that sequence.

2.2 Turing's proof

Consider that the set of all *TMs* is enumerable (namely, it can be put in one-one correspondence with the set of natural numbers). In the first place, any *TM* can be identified with its machine table. A machine table, indeed, consists of a set of quintuples drawn from a finite vocabulary, and the set of all symbol strings which can be built on this vocabulary is enumerable. In the second place, any machine table may be seen as a single symbol string made up by concatenating its quintuples. Hence, the set of all machine tables is enumerable as well, because it is a decidable subset of the set of all symbol strings.

Once we have the list of all possible *TMs* at hand, we can assign each of them a *description number*, namely a number, obtained through a process named *Gödelization*,[20] that completely represents a specific machine table.

Now we have all the notions needed to understand Turing's negative solution of the decision problem. Assuming Turing's Thesis, as specified at the end of §1.2, his negative solution of the decision problem derives from the preliminary solution of another more specific problem, known as the *satisfactoriness problem*, which is a type of halting problem: Is it possible to find a *TM* that decides whether an arbitrary *TM* is circle free?

Once he has demonstrated that the satisfactoriness problem is unsolvable, Turing considers a second problem, called *printing problem:*[21] Is it possible to find a *TM* that decides, in a finite number of steps, whether an arbitrary *TM* will eventually print a 0 on the tape?

Turing then proves that, if the printing problem is solvable, the satisfactoriness problem is solvable as well. But we already know that the latter is unsolvable. Then we conclude (for *modus tollens*), that the printing problem is unsolvable as well.

The last step towards the solution of the decision problem concerns the translation of any *TM* in formulae of a first-order language. This way, it is possible

[19] More precisely, Turing takes the sequence computed by a machine to represent *any* real number whose decimal part, expressed in binary, is equal to that sequence. To avoid confusion, then, he focuses in his 1936 paper on computable sequences, rather than computable numbers (A. M. Turing, *On computable numbers*, cit., p. 233).
[20] See G. S. Boolos, J. P. Burgess, R. C. Jeffrey, Computability and logic, Cambridge University Press, Cambridge, England (2002), § 4.1.
[21] The following explanation is drawn from A. Wells, *Rethinking cognitive computation: Turing and the science of the mind*, Palgrave McMillan, Basingstoke, 2005, Ch. 15.

to interpret the computational steps of a *TM* as subsequent steps in a first order logic proof. Turing is then able to produce a first order formula, *Un*(*M*), which expresses the following proposition: 'In some complete configuration of *M*, 0 is written on the tape'. It is thus possible to prove that the following implications are true:

(a) if, in some complete configuration of *M*, 0 is written on the tape, then *Un*(*M*) is provable in first order logic;
(b) if *Un*(*M*) is provable in first order logic, then, in some complete configuration of *M*, 0 is written on the tape.

The conjunction of a and b tells us that *Un*(*M*) is provable in first order logic if and only if *M* will write 0 on the tape; but we already know that the printing problem is unsolvable. The conclusion is that no algorithm can determine whether a specific first-order statement, namely *Un*(*M*), is provable in first order logic. But this entails that no general algorithm exists that can decide, for any first order statement, whether it is universally valid or not. Therefore, the decision problem is unsolvable.

It is important to note that, even though Turing's negative solution of the *Entscheidungsproblem* is the result of a formal proof, his argument ultimately relies on the intuitive correspondence between the computational processes of a *TM* and his analysis of human computation, that is to say, on the intuitive justification of Turing's Thesis.

3. The Cognitive Relevance of Turing's Work

We have seen in the last section a rigorous formal proof which is ultimately based on the intuitive correctness of the underlying analysis of human cognitive powers and limits. The fact that a mathematical proof could be based on intuitions, however, should not surprise the reader. The link between mathematical proofs and cognitive performances has been highlighted, *e.g.*, by Giuseppe Longo,[22] who notes that the notion of *order*, on which a large part of mathematical reasoning relies, is founded on cognitive notions such as the *mental number line*, through which we can internally 'observe' the geometrical relations between numbers.

Turing explicitly recognizes the importance of a serious cognitive analysis for the faithful formal characterization of an intuitive notion. However, this point represents only an indirect contribution to the study of cognition. We propose that Turing's work may be thought of as an attempt to characterize a specific class of cognitive phenomena, namely, the *phenomena of human computation*. In this section we present, first, Wells' interpretation of Turing's theory of computation, which considers Turing machines as models of mind-

[22] G. Longo, *Reflections on Concrete Incompleteness*, «Philosophia Mathematica», 19, 3, 2011, pp. 255-280 (p. 256).

environment interaction.[23] Second, we propose a methodological interpretation of Turing's Thesis that suggests a precise line of research in cognitive science, according to which *algorithmic skills* are seen as a set of cognitive capacities that deserve to be studied on their own.

3.1 TMs as models of mind-environment interaction

A long established tradition in cognitive science considers Turing as a precursor of *classic computationalism*, according to which cognition is seen as the result of *purely internal* rule-based symbolic transformations. The major reasons for ascribing Turing's thought to this theoretical framework are fundamentally two: 1) classic computationalism considers the *digital computer* as a model of the mind, and the ideal prototype of a digital computer is the *universal* Turing Machine;[24] 2) Turing implicitly advocated the *computational theory of mind*,[25] a basic tenet of classic computationalism, in his famous 1950 article, published in the philosophical journal *Mind*, where he proposed the *imitation game* as a test for machine intelligence.[26] There, he pointed at digital computers as good candidates for playing this game, and was confident that machines of this kind would eventually pass the test, thus showing intelligence and thought. But, as Turing claimed, a digital machine would pass the test in force of the right kind of algorithms it would execute. Thus, intelligent thought processes would be nothing over and above algorithmic activities.

Nevertheless, the ascription of Turing's thought to classic computationalism is marked by a fundamental misunderstanding, which consists in considering the external tape of a *TM* as an *internal* memory.[27] This misinterpretation leads to a view of cognition in which all cognitive activities are made up of rule based transformations of *internal symbols*, which encode either the mental or the environmental features relevant to a given cognitive task. But we have seen, on

[23] *Op. cit.*; A. Wells, *Turing's analysis of computation and theories of cognitive architecture*, «Cognitive Science», 22, 1998, 269–294.
[24] A. M. Turing, *On computable numbers*, §6. A universal *TM* is a *TM* that can emulate the work of any other *TM*. To this purpose, the tape of the universal *TM* is split into two parts by a symbol that is never canceled or overwritten during the computation. On one part of the tape (say the right one), the machine table of the *object TM* (i.e., the *TM* to be emulated) is encoded, while on the other part the universal *TM* keeps a representation of the other components of the object *TM* (the internal and the external memory with the respective contents, and the position of the external head). The analogy with a digital computer is straightforward: the machine table of the universal *TM* corresponds to the *hardware* of a digital computer (specifically, to its CPU), while the quintuples of the object *TM,* encoded on the right part of the tape, correspond to the *software*, namely, to the program in use.
[25] The computational al theory of mind claims that all thought processes have an algorithmic nature.
[26] A. Turing, *Computing machinery and intelligence,* «Mind», 59(236), 1950, 433-460.
[27] See A. Wells, *Rethinking cognitive computation: Turing and the science of the mind*, Basingstoke 2005, for a theoretical analysis of this misunderstanding and its consequences for the *status* of Turing's theory of computation with respect to the philosophy of mind and the philosophy of cognitive science.

the contrary (§1.2), that the tape of a *TM* is an abstraction of 'a child's arithmetic book', so that it is to be considered as a part of the *external environment* used to carry out the cognitive task. By a cognitive point of view, then, a computation is not just a sequence of transformations of symbols of the internal vocabulary of the machine, but it consists in operations on *external symbols*, namely, on the symbols of the vocabulary *of the tape*. These symbols represent environmental features that need not be internalized in order to carry out the cognitive task, because in most cases their internalization would only result in wasting cognitive resources.

If we consider the *mind as a computer* analogy from this perspective, and take a universal *TM*, instead of a digital computer, as a model of cognition, the resulting view turns out to be very different from classic computationalism. For, now, not only symbols (as shown above), but also *programs* need not be internalized in order to be executed. This is tantamount to saying that a cognitive system may not keep the rules of behavior it needs to survive in an internalized library of programs. On the contrary, the external world itself may supply the cognitive system with the symbolic structures needed to adapt its behavior to a continuously changing environment. From this perspective, then, Turing's cognitive view can no longer be seen as a precursor of classic computationalism, but it is more akin to recent approaches to cognitive science such as embodied/extended cognition and the dynamical approach to cognition.

3.2 The methodological interpretation of Turing's Thesis

Wells' interpretation of Turing's cognitive view, on which the previous section is based, expresses an original position in the philosophy of mind (named *ecological functionalism*) which is relevant to the debates on classic *vs* new cognitive science[28] and on the *status* of the computational theory of mind.[29] However, it is not clear how to use this interpretation of a *TM* as a general model of cognition to actually study real cognitive phenomena. To this purpose, we need a step back to the motivations underlying Turing's choices for the design of his computational models.

As we have seen, Turing's proof (§2.2) is based on an analogical argument, namely, it presupposes that the simple operations carried out by a *TM* are an idealization of those performed by a human being that executes an algorithm with the exclusive aid of paper and pencil. We then proposed the following formulation of Turing's Thesis: *Turing machines are adequate ideal models of human computation.* But how could we define the *type* of cognitive phenomenon of which *TMs* are ideal models? Let us call it *phenomenon of human computation*. By this expression we mean any activity of a human being that consists in

[28] A. Newen, L. De Bruin, S. Gallagher, (Eds.) *The Oxford handbook of 4E cognition*, Oxford University Press, New York (NY) 2018.
[29] R. A. Wilson, *Wide computationalism*, «Mind», 103(411), 1994, 351-372; G. Piccinini, *Computationalism in the philosophy of mind*, «Philosophy Compass», 4(3), 2009, 515-532.

the deliberate and controlled execution of a given algorithm, i.e., a purely mechanical or effective procedure. Any specific instance of a phenomenon of human computation is thus uniquely individuated by its particular task, which consists in the request of executing a specified algorithm. An algorithm (or a mechanical or effective procedure) is a finite set of clear-cut formal instructions for symbol manipulation. Given a finite collection of initial data, a human being must be able to carry out such instructions in a definite sequence of steps, with the exclusive aid of paper and pencil (or similar external supports or devices), and without resorting to any special insight or ingenuity.

We have also seen that a *TM* is a *highly idealized* model of a human computer that, given these idealization, is not *per se* fit to model the *whole* variety of phenomena of human computation. In fact, standard *TMs* are able to model only a small class of this kind of cognitive phenomena, namely, those that can be carried out on a unidimensional tape. This means that all the algorithms that need a bidimensional paper (like the standard column algorithms for solving the four basic arithmetical operations) could not be modeled by *TMs*. And we would even need one more dimension to solve, for example, a Rubick's cube, or any other cognitive task involving the execution of some algorithm on a three-dimensional support. If we want to design a model of the full set of phenomena of human computation, we then need to somehow soften these idealizations by retaining the basic *cognitive* insight of Turing's analysis.

To sum up, we can thus formulate the following *methodological interpretation* of Turing's Thesis: *The kind of model adequate to describe and explain all phenomena of human computation is to be searched as an appropriate generalization of Turing's machines, which preserve their structure and basic design.*

3.3 Dynamical models of human computation

In §3.1 we have seen an interpretation of *TMs* as models of mind-environment interaction, where the tape of a *TM* is seen as the external environment on which symbol transformation is performed. Taking a cue from this view, we propose an interpretation of *TMs* as idealized *dynamical models* of human computation. Turing himself, indeed, individuated the three main components of these dynamical models:[30] the internal state, the position of the external head and the content of the tape. In order to fulfill the methodological interpretation of Turing's Thesis, then, we propose the definition of an *Enhanced Turing Machine* (*ETM*) as a dynamical model of the full variety of the phenomena of human computation.

An *ETM* is a dynamical model which satisfies the following three requirements:

[30] «We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer». (A. M. Turing, *On computable numbers*, p. 250).

1. Its instantaneous state has the three components individuated by Turing: *q* = internal state, *p* = head position, *c* =content of the external support;

2. its basic actions should be the three ones also individuated by Turing:[31] (i) the head reads the whole content of a finite neighborhood (*read/write neighborhood*) of its present location, but then changes the symbol of just one cell of this neighborhood; (ii) the head moves from its present location to a new one within a fixed finite neighborhood (*move neighborhood*) of the present location; (iii) the machine changes its internal state;

3. finally, the global transition function of an *ETM* should satisfy the constraint that Turing pointed out:[32] At each time step, the next instantaneous state of the machine should be completely determined by the present internal state and by the present content of the read/write neighborhood (*scanned content*), by means of the three basic actions. More precisely, this means that the present internal state and the scanned content should determine (i.a) the cell of the read/write neighborhood whose present symbol is to be changed and (i.b) the new symbol to be substituted for the present one in such a cell; (ii) the cell of the move neighborhood where the head is going to relocate; (iii) the new internal state.

These three requirements represent, in our view, the distillation of the cognitive insight on which Turing's theory of computation is based. A model satisfying these requirements is thus a good candidate for an adequate description and explanation of the phenomena of human computation in all their variety. We proposed elsewhere[33] a dynamical theory of human computation based on a type of dynamical model (named *Algorithmically enhanced Turing Machine – ATM*) that satisfies the above requirements, but we cannot give here the details of these models. In the present paper, our aim has been to highlight the cognitive foundation of Turing's theory of computation and how it is relevant not only to the history of the concept of algorithm, but also to the philosophy of cognitive science and to the scientific study of a *specific type* of cognitive phenomena.

Simone Pinna
Università degli studi di Cagliari
✉ simonepinna@hotmail.it

Marco Giunti
Università degli studi di Cagliari
✉ giunti@unica.it

[31] See footnote 13 of the present paper.
[32] «The operation actually performed is determined [...] by the state of mind of the computer and the observed symbols. In particular, they determine the state of mind of the computer after the operation is carried out» (A. M. Turing, *On computable numbers*, cit., p. 251).
[33] M. Giunti, S. Pinna, *Toward a dynamical theory of human computation*, «Logic Journal of IGPL», 24 (4), 557-569.

References

Primary Sources

Church, A. 1936. *An unsolvable problem of elementary number theory*, «Journal of Mathematics», 58, pp. 345-363.

Giunti, M., Pinna, S. 2016. *Toward a dynamical theory of human computation*, «Logic Journal of IGPL», 24 (4), 557-569.

Gödel, K. 1931. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, «Monatshefte für Mathematik und Physik», 38, pp. 173-198.

Gödel, K. 1934, *On Undecidable Propositions of Formal Mathematical Systems*, in Davis, M. (ed.) 1965. *The Undecidable*, Raven Press, Hewlett, New York, pp. 39-74.

Turing, A. M. 1936. *On computable numbers, with an application to the* Entscheidungsproblem, «Proceedings of the London Mathematical Society» (pp. 230–265), London, Oxford Journals.

Turing, A. M. 1950. *Computing machinery and intelligence,* «Mind», 236)59), pp. 433-460.

Wells, A. 1998. *Turing's analysis of computation and theories of cognitive architecture*, «Cognitive Science», 22, 269–294.

Wells, A. 2005. *Rethinking cognitive computation: Turing and the science of the mind*, Palgrave McMillan, Basingstoke.

Secondary Literature

Boolos, G. S., Burgess, J. P., Jeffrey, R. C. 2002. *Computability and logic*, Cambridge University Press, Cambridge, England.

Dalla Chiara, M. L. 1974. *La logica*, Isedi, Milano.

Longo, G. 2011. *Reflections on Concrete Incompleteness*, «Philosophia Mathematica», 19, 3, pp. 255–280.

Newen A., De Bruin L., Gallagher S. (Eds.) 2018. *The Oxford handbook of 4E cognition*, Oxford University Press, New York (NY).

Piccinini, G. 2009. *Computationalism in the philosophy of mind*, «Philosophy Compass», 4(3), pp. 515-532.

Wilson, R. A. 1994. *Wide computationalism*, «Mind», 103(411), pp. 351-372.