

The X -th Hilbert problem

Massimiliano Berti, Rocco Tarchini

Abstract We present the X -th Hilbert problem proposed by Hilbert himself in the 1900 world mathematician congress, concerning the “mechanical”, or “algorithmic”, solvability of any diophantine equation. We first define the formal notion of algorithm, namely of “general recursive” function, and we discuss its Turing characterization. Then we prove the unsolvability of the “halting problem” and the existence of a recursively enumerable set (i.e. generated by an algorithm) whose complementary set is not recursively enumerable. In light of these results we finally discuss a Gödel incompleteness theorem and, via the DMPR arithmetic theorem, we prove the unsolvability of the X -th Hilbert problem.

1 Introduction: the notion of algorithm

1.1 Diophantine equations

A diophantine equation is an equation of the type $p(x_1, \dots, x_k) = 0$ where p is a polynomial with integer coefficients and the solutions (x_1, \dots, x_k) are searched in the natural numbers. For example

$$2x_1 + 3x_2 = 1$$

with $x_1, x_2 \in \mathbf{N}$ is a diophantine equation generated by a polynomial of order 1, and

$$x_1^2 - x_2^2 = 2$$

by a polynomial of order 2. These equations arise in many mathematical contexts.

Any diophantine equation of order 1 can be solved by the so called euclidean algorithm taught at school. This is a “mechanical” procedure. A creative effort of mathematicians has allowed to discover a general procedure to solve also any diophantine equation of order 2. What about equations of higher order? The answer to this question is very difficult and involves many advanced tools of number theory.

1.2 The Hilbert problem

In 1900 Hilbert made the following famous affirmation in his presentation of twenty-three problems facing mathematics in the future:

“Take any definite unsolved problem such as [...] the existence of an infinite number of prime numbers of the form $2^n + 1$. However unapproachable these problems may seem to us and however helpless we stand before them, we have nevertheless the firm conviction that their solutions must follow by a finite number of purely logical processes. [...] We hear within use the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no *ignorabimus*.”

What are these “purely logical processes”? In the last decades of the nineteenth century, Frege and Peano had showed that the “ordinary reasoning” used for the proofs of mathematical theorems amount to formal manipulations of a finite number of symbols (like quantifier \forall , \exists , logical connectives) acting on a finite alphabet (e.g. the natural numbers). This is the so called formalization of arithmetics. The Hilbert “purely logical processes” are exactly these formal (mechanical) manipulations of symbols. Can we prove all the arithmetics via such a “formal game”?

More precisely, Hilbert proposed the following definite problem in arithmetics: given a diophantine equation with any number of unknowns, devise a mechanical process which determines by a finite number of operations whether the equation is solvable in integers.

In other words, Hilbert asked to solve diophantine equations via an algorithm, that is a “deterministic”, “computable”, “effective” procedure which must be expressed by a finite set of instructions of finite size.

Despite to Hilbert’s guess, such a problem cannot be solved. It has required a very long and difficult conceptual itinerary. We shall present the main steps in the following.

The first thing to do is clearly to precisely define the concept of “algorithm”. Several definitions have been given by Gödel-Herbrand, Kleene, Church, and Turing, see [9], [6], [10]. Happily it has been proved that all coincide. We shortly (but rigorously) present in the Appendix the Turing characterization which has the advantage to be the most intuitive, especially nowadays that we are used to standard computer machines.

For the reader who does not wish to learn the mathematical details, it is sufficient to think to a Turing Machine (TM) as a standard computer program acting on an idealized computer with an infinite memory storage, working with a finite number of digits, and which never breaks down.

Therefore given an input value $x \in \mathbf{N}$, the computer starts performing its instructions. It can be that such computations stop giving an output value $y \in \mathbf{N}$. However, it could also happen that a computer program does not stop when initialized with a certain input value x . For example, it is familiar that sometimes a computer program “enters in loop”, repeating periodically the very same instructions.

In any case any TM defines a function $y := f(x)$ which could not be defined for all $x \in \mathbf{N}$. This is the class of the so called “general recursive functions”.

It is fundamental for the understanding of the following to know that the class of all the general recursive functions is countable, because it is produced by a finite number of instructions of finite size. Moreover the class of all the “general recursive functions” can be “effectively” enumerated as a list

$$f_n(\cdot), \quad n = 1, 2, \dots,$$

namely there is an “algorithmic” manner to write down such a list, see lemma 6.1.

2 The Halting problem

The fundamental problem is the following:

- Is there an “effective” procedure such that, given n and x we can determine whether $f_n(x)$ is defined? In other words, is there an algorithm to decide if the n -th Turing machine, applied to an input x , stops, giving an output $f_n(x)$?

We shall say that the n -th Turing Machine¹ applied to the input value $x \in \mathbf{N}$ is convergent if it stops, giving an output value $f_n(x)$ (“halting” means “having an output”). On the contrary we say that TM is divergent.

Theorem 2.1 *There is no recursive function (of 2 variables) such that*

$$g(n, x) := \begin{cases} 1 & \text{if } f_n(x) \text{ is convergent} \\ 0 & \text{if } f_n(x) \text{ is divergent.} \end{cases} \quad (1)$$

PROOF. It is again based on a diagonal argument. Suppose by contradiction that such a function g does exist. Then we can define the new function

$$\Psi(n) := \begin{cases} 1 & \text{if } g(n, n) = 0 \\ \text{not defined} & \text{if } g(n, n) = 1. \end{cases} \quad (2)$$

The function Ψ is a partial recursive function (of one variable). Indeed $g(n, n) \in \{0, 1\}$ is algorithmically defined for every n .

Being the function Ψ a partial recursive function, by lemma 6.1 there exists $n_0 \in \mathbf{N}$ such that

$$\Psi(n_0) = f_{n_0}(n_0).$$

Then $\Psi(n_0) = f_{n_0}(n_0)$ which, by (2), is convergent (with value 1) if and only if $g(n_0, n_0) = 0$. On the other hand, by (1), $g(n_0, n_0) = 0$ if and only if $f_{n_0}(n_0)$ is divergent. This contradiction proves the Theorem. ■

Let us understand properly the previous result. The list of the pairs (n, x) such that the n -th TM applied to the input x stops DOES exist; as well as the list of the pairs (n, x) such that the n -th TM applied to the input x does not stop. The non existence theorem 2.1 states the impossibility of building effectively such lists via an algorithmic procedure. In other words: God surely knows such lists; but no machine can produce them.

3 Recursive and enumerable recursive sets

Definition 3.1 *We say that a set $S \subset \mathbf{N}^k$ is recursively enumerable if it is the range of a partial recursive function. In other words if S is the set of the outputs of a Turing Machine.*

Definition 3.2 *We say that a set $S \subset \mathbf{N}^k$ is recursive if and only if the “characteristic function” of the set S defined by*

$$\chi_S(n) := \begin{cases} 1 & \text{if } n \in S \\ 0 & \text{if } n \notin S \end{cases}$$

is total general recursive. Equivalently, both S and the complementary S^c are recursively enumerable.

¹We identify via Lemma 6.1 the n -th TM with the function f_n that it produces.

We can now prove the following fundamental theorem

Theorem 3.1 *There exists a set $K \subset \mathbf{N}^2$ which is recursively enumerable yet not recursive.*

PROOF. Define the function

$$\Phi(n, x) := \begin{cases} (n, x) & \text{if } f_n(x) \text{ is convergent} \\ \text{divergent} & \text{if } f_n(x) \text{ is divergent.} \end{cases}$$

This function Φ is partial recursive. Indeed, given (n, x) we can find algorithmically the function f_n (see lemma 6.1) and compute $f_n(x)$. If $f_n(x)$ is convergent then we print (n, x) which is the output of the algorithm Φ . Otherwise, if $f_n(x)$ is divergent then also $\Phi(n, x)$ is divergent.

Then $K = \text{range}(\Phi)$ is recursively enumerable (by definition). But its characteristic function χ_K is equal to the function g defined in (1), which is not a general recursive function by the Halting Theorem 2.1. ■

Remark 3.1 *By Theorem 3.1 we immediately get the existence of a subset $K \subset \mathbf{N}$ which is recursively enumerable yet not recursive, by a computable bijection $\mathbf{N}^2 \leftrightarrow \mathbf{N}$.*

4 Gödel incompleteness and X -th Hilbert problem

Let \mathcal{F} be a formal system, that is a finite alphabet of symbols, a finite set of axioms and of rules of inference, see [8] for a wider discussion. We shall always consider “consistent” formal systems, in which it is not possible to deduce simultaneously both a proposition and its negation (in other words, it is not possible to prove contradictions from the axioms). We shall consider systems in which each formally provable proposition is actually true. We shall say that a formal system is “complete” if every true proposition of the system admits a formal proof within the formal system.

Computability theory provides a perspective from which it can be seen that incompleteness is a pervasive fundamental property of formal systems.

We need the following preliminary result.

Lemma 4.1 *The set of all formally provable propositions in \mathcal{F} is recursively enumerable.*

PROOF. We can algorithmically list all the formal propositions in \mathcal{F} (for example using a lexicographical ordering). Next we can algorithmically check whether each formal proposition obeys the formal rules of inference in \mathcal{F} . Then we have a purely mechanical test to know if each formal proposition is a theorem, see [8] chapter 4. ■

We now consider the formal system of Peano Arithmetics (PA). Let K be the set defined in theorem 3.1 (and remark 3.1). The statement $n \notin K$ is a proposition of this formal system, say P_n .

Theorem 4.1 (Incompleteness Theorem) *There is n_0 such that the statement*

$$n_0 \notin K \quad \text{is true}$$

but P_{n_0} is not provable in the arithmetic formal system (PA).

PROOF. By contradiction suppose that, $\forall n$,

$$n \notin K \text{ is true} \iff P_n \text{ is provable in the formal system.} \quad (3)$$

We now prove that K^c is recursively enumerable, contradicting the fact that K is not recursive, see theorem 3.1. If $n \in K^c$ then the statement $n \notin K$ is true, and, by (3), the proposition P_n is formally provable. But the set of all the formally provable propositions (theorems) is recursively enumerable. ■

We can now consider properly the original X -th Hilbert problem concerning the formal system of Peano arithmetics. After the achievements of Turing, the complete solution of the unsolvability of the X -th Hilbert problem required a long time. The main breakthrough was the following theorem due to joint works of Y. Matiyasevich, J. Robinson, M. Davis and H. Putnam, see [3], [7].

Theorem 4.2 ([3],[7]) *Let $S \subset \mathbf{N}$ be a recursively enumerable set. Then there exists a polynomial $p(n, x_1, \dots, x_k)$ with integer coefficients such that the equation*

$$p(n, x_1, \dots, x_k) = 0 \text{ has integer solutions} \iff n \in S. \quad (4)$$

We underline that the construction of the polynomial p is completely explicit. Let p_0 be the polynomial provided by theorem 4.2 when $S = K$ is the set defined in theorem 3.1. By theorem 4.1 we deduce the following Gödel type result in arithmetics.

Theorem 4.3 (Diophantine Incompleteness Theorem) *There is n_0 such that the equation*

$$p_0(n_0, x_1, \dots, x_k) = 0$$

has no integer solutions, although such a statement is not formally provable in the arithmetics.

Finally, let us conclude proving the negative answer to the X -th Hilbert problem.

The set K is not a recursive set by theorem 3.1. Then, given $n_0 \notin K$, it is not possible to determine algorithmically that n_0 actually is not in K . Therefore, by (4), we cannot determine algorithmically if the equation

$$p_0(n_0, x_1, \dots, x_k) = 0 \text{ has or not integer solutions.} \quad (5)$$

We have found a class of diophantine equations for which it is not possible to decide algorithmically whether they admit integer solutions.

Remark 4.1 *We proved the previous unsolvability result for the PA formal system. Working within the stronger formal system of Zermelo-Frankel (ZFC) for set theory plus the axiom of choice, maybe the statement (5) could be proved. However there exists another $n_1 \notin K$ such that a formula like in (5) is not provable in ZFC, see e.g. [2].*

5 Philosophical conclusion

To develop some conclusions on the philosophical status of Mathematica, we start by the following consequences of the previous results:

At any time T there is a diophantine equation E such that no human being at time T can be sure to be able to answer the question whether E is solvable or not.

Indeed, at any time T , there exist no algorithm to generate K^c . Therefore, for a $n_0 \in K^c$, we shall never possess an algorithm which is able to give us the answer: “ $n_0 \in K^c$ ”. Via some more powerful mathematical reasoning of our mind, we might have the “insight” on which are the numbers of K^c . But we are not sure to receive such a light! Therefore, at any time T there exist n_0 such that I’m not sure to be able to answer: “the polynomial equation $p_0(n_0, x_1, \dots, x_k) = 0$ has not solutions”.

Then, how can Mathematics be an a-priori of the human mind? In the ”Critic for pure reason” Kant argues that...

6 Appendix

6.1 The primitive recursive functions

The following class of functions, that we shall call the “primitive recursive functions” (p.r.f.), must be considered “algorithmic”, “effectively computable”.

- (i) The constant functions $f(x_1, \dots, x_k) := c$.
- (ii) The successor function $f(x) := x + 1$.
- (iii) The identity functions $f(x_1, \dots, x_k) := x_i, i = 1, \dots, k$.

Then we can produce new recursive functions via “composition” and “induction”:

- (iv) Composition of p.r.f. yields a new p.r.f.
- (v) Induction

Deterministic operations like “sum” and “multiplication” are primitive recursive functions as well as every polynomial.

However, it is possible to construct functions, intuitively computable by an algorithm, which are not primitive recursive. For example let us consider the following function of three variables:

$$\begin{aligned} f(0, x, y) &= y + x \\ f(1, x, y) &= y \cdot x \\ f(2, x, y) &= y^x \\ &\dots = \dots \end{aligned}$$

$f(z + 1, x, y) =$ result of applying y to itself $(x - 1)$ times under the $z - th$ level operation.

It can be proved that such a function cannot be produced by compositions of primitive recursive functions, see [9]. However it is universally accepted as a computable function.

In conclusion, the class of primitive recursive functions does not exhaust the concept of “effectively computable” functions.

6.2 The general recursive functions

At first glance an insuperable obstacle to give a definition of “effectively computable” functions (that we shall call “general recursive” functions) appears. Indeed, suppose one had an “effective” definition of general recursive functions. Since there are only countably many instructions one can give for performing a calculation, one could enumerate these functions, say $f_n(x)$, $n = 1, 2, \dots$. Then we can define the new function

$$g(n) := f_n(n) + 1. \quad (6)$$

Such a function g is clearly produced by an algorithmic procedure. Hence g belongs to the previous list of general recursive functions, i.e.

$$\exists n_0 \text{ such that } g(\cdot) = f_{n_0}(\cdot). \quad (7)$$

But, let us consider the value $g(n_0)$. On one side, by (7), $g(n_0) = f_{n_0}(n_0)$. On the other side, by (6), $g(n_0) = f_{n_0}(n_0) + 1$. Such a contradiction proves that an “effective” definition of general recursive functions does not exist.

Remark 6.1 *The previous procedure is an example of the “Cantor diagonal argument” that we shall encounter several times in the following.*

The way out of this dilemma is to give a definition of “general recursive” functions which are not defined for all the values $x \in \mathbf{N}$. We shall call such functions “partial recursive” functions.

Let us make some examples.

Example 1. Consider the function $f(x)$ defined as follows: carry out the decimal expansion of π until a run of at least x consecutive 5 appears; if and when this occurs define $f(x)$ as the position of the first digit of this run. We do not know, at the moment, whether this algorithmic procedure produces a function defined for all $x \in \mathbf{N}$. Such f is then a partial recursive function.

Example 2) Consider the function $f(x)$ defined as follows: examine successive even numbers greater than 2 until one appears which is not the sum of two primes; if and when this occurs give the output $f(x) = 0$. Also in this case we ignore whether $f(x)$ is defined for all $x \in \mathbf{N}$. It is a famous mathematical conjecture that this holds true (Goldbach conjecture). However nobody knows, up to now, the answer.

For partial general recursive functions, the previous diagonal argument does not lead to a contradiction because $g(n_0)$ does not need to have a value, namely n_0 could not belong to the domain of definition of the function g .

Several definitions of partial general recursive functions have been given by Gödel-Herbrand, Kleene, Church, and Turing, see [9], [6], [10]. Happily it has been proved that all give rise to the same class of functions. We shall follow below the Turing characterization which has the advantage to be the most intuitive, especially nowadays that we are used to standard computer machines.

6.3 The Turing machines

A Turing machine is an idealized universal computer device which can operate without restrictions of memory storage and space.

It can be thought as a box, containing a finite set of internal states q_0, q_1, \dots, q_k , together with an infinite tape. The Turing machine (TM) can perform the following basic operations which depend on the internal states and on the input value read in the cell tape being examined:

- (i) A TM can write “1” on the cell of the tape it is examining if there is no yet “1” printed.
- (ii) A TM can erase “1” on the cell of the tape it is examining if there is “1” printed.
- (iii) A TM can shift its attention one cell to the right, or to the left, along the tape.

More precisely a TM can be defined as a finite set of ordered quadruples consisting of symbols for:

- (i) an internal state
- (ii) a possible tape cell input value
- (iii) an operation
- (iv) an internal state.

Such a quadruple expresses an instruction for a TM: given the internal state (i) and the tape cell input value (ii), the machine performs the operation (iii) and takes the internal state (iv) (this set of instructions can be thought as a computer program).

A TM is a deterministic device, that is, given an internal state (i) and a tape cell input value (ii), the operation performed in (iii) is uniquely determined (there are not 2 different quadruples having the same two symbols (i) and (ii)).

Now we can define the following class of functions $y = f(x)$ of one integer variable. Given $x \in \mathbf{N}$, build a tape with x consecutive 1 digits (and 0 outside). This tape is the input for a specific TM which has the initial internal state q_0 . The TM examines first the cell containing the first 1 digit. Then the TM performs its calculus and if and when it stops take as $y := f(x)$ the number of digits 1 occurring in the final tape. In this case any TM defines uniquely a function

$$f : \mathcal{D}(f) \subseteq \mathbf{N} \rightarrow \mathbf{N}$$

which is properly defined only on a subset $\mathcal{D}(f) \subseteq \mathbf{N}$. If, given $x \in \mathbf{N}$, the calculus of the TM never stops, we cannot define the value $f(x)$. The class of all the functions produced by a TM is called the class of the “partial recursive functions”.

It is clear that the previous definition of “partial recursive functions” can be extended to functions with more than one integer variable.

Remark 6.2 *Recalling the example 1), it is clear that it is possible to write a TM (computer program) to check if, given any x , there exists a run of x -consecutive 5 in the decimal expansion of π . We do not know if, given x , such a program stops, providing an answer. Similarly for example 2.*

Remark 6.3 *There are TMs which, for any $x \in \mathbf{N}$, never stop (in such a case they do not define any function). Let us show an example of such “inefficient” Turing Machine. The TM defined by the very simple instruction q_011q_0 never stops. Indeed the previous set of instructions define a “loop”: such a machine keeps on reading and printing indefinitely the value 1 on the same cell tape.*

It can be easily recognized that one can define Turing Machines to perform primitive operations like sum, multiplication, composition, etc... (that is all the primitive recursive functions). We refer to [9] for an explicit example of a TM machine programmed to calculate the function $x \mapsto 2x$. With a bit of expertise one is led to define a function to be “effective”, “computable”, “effectively computable”, “recursively computable”, “mechanically computable”, “produced by an algorithm”, if it is defined by a Turing Machine.

The following lemma is very important for the following.

Lemma 6.1 *The class of the partial recursive functions is countable and can be “effectively” enumerated as a list f_n , $n = 1, 2, \dots$*

PROOF. The set of Turing Machines is countable, because it is produced by a finite number of instructions of finite size. We can arrange all the Turing Machines through a lexicographical ordering, listing first the TM’s defined by 1 quadruple, next the TM’s defined by 2 quadruples, by 3 quadruples, and so on. Such a procedure is purely algorithmic.

■

Remark 6.4 *There exist functions $f : \mathcal{D} \subseteq \mathbf{N} \rightarrow \mathbf{N}$ which are not partial recursive. Indeed the class of the partial recursive function is countable. On the other hand the class of all functions $f : \mathcal{D} \subseteq \mathbf{N} \rightarrow \mathbf{N}$ is not countable. This is a theorem due to Cantor and it is based on a diagonal argument as above.*

The claim that the Turing characterization of “partial recursive functions” grasps the essence of the notion of “algorithmic procedure” is called the “Church’s Thesis”. The strongest argument to support it is that all the independently given formal characterizations of the intuitive notion of “algorithm” -provided by Church, Gödel, Kleene, Turing- have been proved to be equivalent, see [9], [6], [10].

In the following we shall accept the Church’s Thesis as granted.

We now prove an analogous unsolvability result: there exists no algorithm which selects all the TM which stop for every input $x \in \mathbf{N}$.

Theorem 6.1 *There is no effective procedure for deciding, given any n , whether or not f_n is a total function. That is to say, there is no recursive function g such that*

$$g(n) := \begin{cases} 1 & \text{if } f_n \text{ is total} \\ 0 & \text{if } f_n \text{ is not total.} \end{cases}$$

PROOF. Suppose by contradiction that such g exists. Then we produce algorithmically the list of the functions $f_n(\cdot)$ which are total. Then $\Psi(n) := f_n(n) + 1$ is another total recursive function² which does not belong to the list. ■

²Because composition of general recursive functions gives another general recursive function, and the successor function is a primitive recursive function.

We conclude this section with the following observation due to Penrose [8], chapter 2, which highlights the difference between the power of the human mind and a machine. Theorem 2.1 does not exclude the possibility to construct a more “intelligent” algorithm which sometimes manages to “understand” if the n -th TM applied to the input value x stops. Let define it by

$$H(n, x) := \begin{cases} f_n(x) & \text{if } f_n(x) \text{ is convergent} \\ 0 & \text{if } f_n(x) \text{ is not convergent (in this case } H \text{ “realizes” it)} \\ \text{divergent} & \text{if } f_n(x) \text{ is not convergent (in this case } H \text{ does not “realize” it)} \end{cases} \quad (8)$$

We can define the partial recursive function of one variable

$$\Psi(n) := 1 + H(n, n) \quad (9)$$

and, therefore, by Lemma 6.1, there exists $n_0 \in \mathbf{N}$ such that

$$\Psi(\cdot) = f_{n_0}(\cdot). \quad (10)$$

Let us compute $f_{n_0}(n_0)$ and suppose it is convergent. Then

$$\begin{aligned} f_{n_0}(n_0) &\stackrel{(10)}{=} \Psi(n_0) \stackrel{(9)}{=} 1 + H(n_0, n_0) \\ &\stackrel{(8)}{=} 1 + f_{n_0}(n_0) \end{aligned} \quad (11)$$

which is a contradiction. Therefore $f_{n_0}(n_0)$ is divergent. Furthermore, by (11), we also see that $H(n_0, n_0)$ can not be equal to 0. Otherwise $f_{n_0}(n_0) = 1$, but we just proved that $f_{n_0}(n_0)$ is divergent. In conclusion the only possibility which remains is that both $f_{n_0}(n_0)$ and $H(n_0, n_0)$ are divergent.

We have defeated the algorithm! That is to say we have proved (!) that $f_{n_0}(n_0)$ is divergent, but the machine does not. Indeed $H(n_0, n_0)$ is not equal to 0.

We have applied a type of reasoning which is qualitatively stronger than a mere calculus! The “insight” through which we concluded that $f_{n_0}(n_0)$ is divergent is an example of what the logicians call “reflection principle”. We shall encounter again this idea in the Gödel type Incompleteness Theorems 4.1 and 4.3. In both theorems we shall prove (not algorithmically!) a true statement which any machine will never, ever, achieve!

References

- [1] Cohen P., *Set Theory and the Continuum Hypothesis*, W. A. Benjamin, Inc. New York, 1966.
- [2] Davis M., *The Incompleteness Theorem*, Notices of the American Math. Soc., April, 2006.
- [3] Davis M., Putnam H., Robinson J., *The decision problem for exponential Diophantine equations*, Annals of Math., vol. 74, n. 3, pp. 425-436, 1961.
- [4] Gödel K., *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter systeme*, I, Monatsh. Math. u. Phys. 38, 1931.

- [5] Hilbert D., Ackermann, *Grundzüge der theoretischen Logik*, 3, 1, 1928.
- [6] Kleene S., *λ -definability and recursiveness*, Duke Math. Journal., vol. 2, pp. 340-353, 1936.
- [7] Matiyasevich Y., *Hilbert's X-th problem*, MIT Press, Cambridge, Massachusetts, 1983.
- [8] Penrose R., *The Emperor's new Mind*, 1992, Oxford University Press, 1989.
- [9] Rogers H. Jr. *Theory of recursive functions and effective computability*, Mc Graw Hill, New York, 1967.
- [10] Turing A. *Computability and λ -definability*, Journal of symbolic Logic, vol. 2, pp. 153-163, 1937.