Studia Semiotyczne—English Supplement, t. XXX (2019), s. 107–127
ISSN 0137-6608, e-ISSN 2544-073X
DOI: 10.26333/stsen.xxx.06
Article

Paweł Stacewicz*

UNCOMPUTABLE NUMBERS AND THE LIMITS OF CODING IN COMPUTER SCIENCE[1]

Summary: The description of data and computer programs with the use of numbers is epistemologically valuable, because it allows to identify the limits of different types of computations. This applies in particular to discrete (digital) computations, which can be described by means of computable numbers in the Turing sense. The mathematical fact that there are real numbers of a different type, i.e. uncomputable numbers, determines the minimal limitations of digital techniques; on the other hand, however, it points to the possibility of the theoretical development and physical implementation of computationally stronger techniques, such as analogue-continuous computation. The analyses presented in this article lead to the conclusion that physical implementations of unconventional (non-digital) computations require the occurrence of actually infinite quantities in nature. Although some arguments of theoretical physics support the physical existence of such quantities, they are not definitive.

Keywords: numerical coding, computable numbers, uncomputable numbers, Turing machine, digital computation, analogue computation, infinity.

* Warsaw University of Technology, Faculty of Administration and Social Sciences. E-mail: p.stacewicz@ans.pw.edu.pl. ORCID: 0000-0003-2500-4086.

[1] I thank two anonymous reviewers for valuable comments and advice which significantly improved the original version of the text. I also thank the professors Witold Marciszewski and Andrzej Biłat for fruitful, editorial discussion of the text. For all errors, ambiguities and shortcomings remaining in the work I take responsibility and I apologize in advance for them to all readers.

From the point of view adopted in this work, computational objects, in particular computer programs, mediate between the mathematical sphere of numbers and physical reality. For example: a sound playing program operates on numerical representations of acoustic waves, and its instructions cause, due to the appropriate design of the computer, real physical vibrations of air molecules. What is more, this and any other program can be analysed on two levels, i.e. as an object of two types: on the one hand as a series of symbols that can be reduced to numbers, and on the other—as a strictly defined system of physical states of the machine (which, after running the program, cause regular changes of its subsequent states).[2] Due to the indicated correspondence, many computer-related issues can be resolved by referring to the properties of numbers—numbers that according to a particular, machine's specific, model of computation (e.g. digital or analogue) correspond to the data, texts and results of the programs.

In this work I shall focus on programs for digital machines. They are described theoretically by means of the Turing model of computation (universal Turing machine), and speaking "numerically", using computable numbers in the sense of Alan Turing. Referring to certain properties of computable and uncomputable numbers, in particular the fact that the digital representations of uncomputable numbers are actually infinite, I shall determine the theoretical reasons for the existence of computational limitations of such programs. I shall also discuss the possibilities of overcoming these limitations by means of computational techniques that (theoretically) allow the processing of signals described using uncomputable numbers in the Turing sense. The presented text is for the most part a review. However, it contains a number of the author's interpretations of the results of computer science and its mathematical foundations research (e.g. results of A. Turing and G. Chaitin), in particular interpretations regarding the infinite nature of uncomputable numbers and codes considered in theoretical computer science.

[2] Some philosophers of computer science speak directly—adopting an ontological rather than epistemological attitude—about the dual, i.e. abstract-physical, nature of computer programs (Moor, 1978; Colburn, 2000; see also Angius & Turner, 2013).

1. Numbers, Computing and Numerical Coding

The most important, and oldest, idea that resulted in the creation of computers and then computer science is the idea of numerical coding.[3] Behind it is the belief that the world of numbers (maybe even only natural ones) and relatively simple operations on them (such as comparing, adding or dividing) is rich enough to represent various aspects of the real world.

In modern computing, *numerical coding,* understood as describing data processed by computers using numbers,[4] is a common and perhaps theoretically necessary activity.[5] It is already present at the level of initial formalization of some tasks, when the objects appearing in these tasks (e.g. text, sound or graphic) are described by means of numbers, specially selected and included in appropriate structures. For example: characters processed by text editors are assigned specific numbers (according to e.g. ASCII code), while images displayed on monitors are often coded in the form of a sequence of numbers that determine the coordinates and colours of points on the raster matrix. At the lowest level of intra-computer structures, the relevant codes are created automatically, thanks to specially designed programs (e.g. compilers). Most importantly, however, in

[3] Its oldest manifestation was probably the philosophy of the ancient Pythagoreans, which postulated reducing all fragments of reality to some kind of numbers (summarized in the short slogan that "everything is a number"). In modern philosophical thinking, especially in the context of computer science philosophy, Pythagorean ideas are revived, which some call neopythagoreanism. This is due to a kind of feedback: Pythagorean ideas contributed to the emergence of computer science, and its successes, among others in the field of simulation of physical phenomena by means of operations on computer-represented numbers, strengthen the Pythagorean view of the world (Krajewski, 2014).

[4] In the computer science context, the term "describing data processed by computers using numbers" usually has a syntactic rather than an abstract sense. This means that it is about coding data using symbolic (and physical) representations of numbers, e.g., zero-one sequences. In the present text I shall also refer to the abstract (strictly mathematical) properties of numbers and their sets, such as the continuity of a set of real numbers. In the case of insufficient context, however, I shall signal whether in the given place it is about the abstract or syntactic dimension of the concept of number (writing e.g. that it is about decimal expansion of a number).

[5] See the online discussion on the Cafe Aleph blog that resulted from the creation of this work (Stacewicz, 2018b).

mathematical terms, aside from the physical design of the computer and the physical processes of signal processing, they can be represented numerically, for example in binary form.

The last five words of the previous paragraph indicate that I understand the term "numerical coding" widely in this work. In particular, I understand it more broadly than the term "digital coding", which I reserve for the way of representing information in digital computers, which are machines with discrete states operating on binary signals. I take the broad term "numerical coding" to be reasonable, because computer science, generally conceived, considers a wider class of machines than the digital. This broader class includes analogue circuits that allow (at least theoretically) operation on continuous signals described by real numbers,[6] as well as quantum computers for which the basic unit of information is the q-bit, mathematically defined using complex numbers.[7]

The concept of numerical coding is closely related to the key computer science concept of *computing*. In the context of problem solving, it means the mechanical implementation of the process of determining the value of the function, which assigns its specific solutions to the input of the problem (solutions for specific data).[8] If the data are numerically encoded, then the arguments and values of this function are those types of numbers (e.g. natural or real), which are allowed by the coding method appropriate for a given machine. This is determined by the appropriate model of computation (e.g. digital or analogue). Let us also say that the computer description, not purely mathematical, of the calculated function

[6] See works by Shannon (1941) and Rubel (1993).

[7] I also use the term "numerical coding" in another work (Marciszewski & Stacewicz, 2011, pp. 75–77). A similar conceptual convention is found in Krajewski, who does not use the term "numerical coding", but distinguishes digitization as one of the types of coding (although the most common), fundamentally different from data coding in analogue circuits processing signals described by real numbers (Krajewski, 2014).

[8] Historically, the first mature considerations for solving problems using calculations (computations), i.e. mechanical operations on physical equivalents of numbers, are due to G.W. Leibniz. For the modern concept of computing, the following ideas and achievements are particularly important: the design of a calculating machine (performing the four basic arithmetic operations), the invention of a binary arithmetic system, the design of a machine operating on binary encoded numbers, as well as the concept of a universal symbolic language (*lingua characteristica*) and coupled with it a reliable calculus (*calculus ratiocinator*). See the work by Trzęsicki (2006).

is either the program text (if the machine accepts programs written in a certain programming language) or the connection diagram between the elementary systems of the machine (if the machine is physically programmed like analogue systems or the first digital computers).

Since the vast majority of today's computers perform digital computations, in the following paragraphs I shall take a closer look at the "numerical" characteristics of the tasks entrusted to them. In particular, I shall consider the question as to whether the numerical codes desired in their description must be finite, or if sometimes it is necessary to refer to the concept of infinite code.[9]

At first glance, all the codes involved are finite, and thus reduced to natural numbers. This suggests the observation that the data entered into the digital computer have a finite representation, and the programs used to process them are finite sequences of instructions that, when encoded in binary form, can be interpreted as natural numbers. A deeper reflection on the functions of digital computers, however, leads to the statement that the theoretical analysis of the capabilities of these computers must refer to the concept of infinite code (even if such codes cannot be implemented inside real digital machines). Two possible contexts of reference should be distinguished.

First, in the case of many real problems (e.g. in the fields of dynamics or mechanics), the results obtained for specific input data can be expressed in *irrational numbers*, those with infinite and irregular expansions (e.g. decimal). This happens, for example, when a given problem is formu-

[9] The concept of infinite code—that is, the result of the coding process that has (actually) infinite length—is a non-standard concept that goes beyond the standard theory of computation, expressed e.g. in terms of Turing machines. However, in modern computer science methodology, which also includes some non-standard models of computation, this concept is used—e.g. to refer to the infinite length of program codes or the infinite tape of the Turing machine, which is completely filled with data (Ord, 2002, p. 17; Ord, 2016, p. 146; Mycka & Olszewski, 2015, pp. 58–59). Let us emphasize, however, that this concept makes sense when one makes an (even working) assumption of the possibility of going beyond the traditional Turing model of computation. The use of the concept of infinite code is justified in the present work, because later in it (especially in section 4) I shall analyse the possibility of the physical implementation of non-Turing computations, also those that include infinitistic elements. Regardless of this intention, in this section I show how (general) analysis of problems that we would like to solve traditionally (i.e. digitally) leads to the necessity of at least a critical consideration of non-traditional (i.e. infinite) codes.

lated mathematically using a certain equation (e.g. differential) and the root of this equation is an irrational number (such as $\sqrt{2}$, π or e). In this case, the result is *de facto* represented by an infinite number. Let us first note, before explaining in more detail in section 3, that the most troublesome situation occurs when we deal with this kind of irrational number, which is uncomputable in the sense of Turing. Secondly, however, and crucial for further analysis, each more complex programming task has an *infinitystic* structure. This means that the set of its initial data, and sometimes also the set of its potential results, is unlimited. As a simple example, let's consider the problem of determining the roots of quadratic equations $ax^2 + bx + c = 0$, where the range of possible a, b, c coefficients to enter is unlimited. In the case of this problem, there is, despite an unlimited field, a finite method of finding the x values sought, which is the commonly known "delta" algorithm. There is also a finite program (many), which for any input data (i.e. a system of coefficients a, b, c) allows, in a finite number of steps, the generation of the correct result. This program must be treated as a general (computer) solution to the problem posed, a solution which corresponds to the finite numerical code of the program (in short: a certain number).[10]

Unfortunately, for other problems with an unlimited input data domain, the numerical code of the general solution—which is a digital record of all possible pairs <INPUT, RESULT>, or in other words, the function that assigns the results—must remain infinite. This happens when there is no finite program to solve the problem. If such a program exists, it is a form of encoding the set of the given pairs in the shape of a procedure that generates correct results (for all possible input data) which is "intelligible" for a digital machine. The code of such a procedure corresponds to a natural number (written e.g. as a sequence of zeros and ones). If such a program does not exist, it must be assumed that the overall solution to the problem corresponds to some uncomputable number in the Turing sense (i.e. a certain special irrational number with infi-

[10] Let us emphasize here that, although the question of the infinite domain of input data may be irrelevant from the point of view of solving the algorithmic task, the fact that this solution applies to an unlimited number of input data determines its strength. It is in a way a universal solution (similar to mathematical theorems, it applies to an infinite number of special cases). In some situations, however, the infinite field can lead to trouble—more on this in the main text (see also Stacewicz, 2015).

nite expansion, which no digital machine can calculate; see further in section 2).

In the context of solving problems by computing that interests us here, infinite numerical codes can, therefore, occur at two levels: 1) at the code level of the exact individual result, 2) at the code level of the overall problem solution. In both cases, it may happen that the appropriate code is in the form of an uncomputable number, and then—as we shall see in section 3—the method sought to solve the problem lies beyond the limits of the possibilities of digital coding (which does not, however, exclude the existence of such a method that would be implemented on machines of types other than digital).

2. Uncomputable Numbers in the Turing Sense

The *uncomputable numbers* highlighted in the title of this article were defined by Alan Turing in his work from 1936 entitled *On Computable Numbers, with an Application to the Entscheidungsproblem.* He defined them as such irrational numbers, whose decimal representation cannot be determined with any given accuracy, by any system for mechanical calculations, today called the Turing machine.[11] In the modern style, we would say that these numbers are indeterminate by means of algorithms for digital machines, and therefore those for which there are no finite computer programs that allow step by step calculation of the subsequent digits of their decimal or other representations (although such representations are strictly defined, see Stacewicz, 2012). For example: the irrational number e does not have the above properties, because it is relatively easy to generate successive digits of its expansion by means of a program calculating successive subtotals of the appropriate series (remember that $e=\sum_{n=0}^{\infty}\frac{1}{n!}=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots$). Therefore, it is not an uncomputable number, although it is characterized by irrationality.

Unlike the irrational number e, uncomputable quantities in the Turing sense are defined in a way that excludes the possibility of their successive approximation using Turing machines or equivalent computational mech-

[11] It is worth adding that Turing first gave the exact definition of a set of computable numbers (numbers whose decimal notation can be determined definitively or with any given accuracy using a finite program for a Turing machine), and then proved the existence of real numbers of another type (see further in main text), or uncomputable numbers (Turing, 1936).

anisms. This is determined by Turing's original reasoning, which, after defining computable numbers, proved that there are real numbers of another type, and then specified a set of non-computable numbers as the complement to the set of computable numbers in the set of real numbers. I shall present this reasoning in a sketchy and demonstrative way—limiting it to real numbers in the range (0,1).[12] The starting point of the argument is that the result of the operation of each Turing machine for specific input data—a machine generating series of digits from the set {0,1, ..., 9}—is clearly represented by a certain real number from the interval (0,1). It is such a number whose decimal expansion is the same as the finite or infinite sequence of digits generated by the machine.

Due to the fact that each machine, together with the input data, unambiguously defines a unique string of symbols (representing its program and the initial content of the tape), each of them can be assigned a unique number, and all machines can be set into an infinitely countable sequence. According to the order in this sequence, you can then set all digit sequences generated by subsequent machines. These sequences form an infinite countable set and unambiguously designate concrete computable numbers in the range (0,1). These are numbers with decimal expansions identical to the subsequent sequences.

Having the above-mentioned sequence list, one can ask if there is such a sequence S on it, that its n-th digit differs (e.g. by 1) from the n-th digit of the n-th sequence on the list (if the n-th sequence is long enough). The postulated S sequence cannot appear in the list because it differs (by at least one number) from each of the sequences in the string. Therefore, it differs from any sequence generated by any machine. Therefore, this sequence must specify a number from the range (0,1) that no machine can generate, i.e. a real uncomputable number (Turing, 1936; Marciszewski & Stacewicz, 2011).[13]

---

[12] In the presented reasoning, Turing skilfully used the diagonal technique, which was used for the first time by G. Cantor in proof of the uncountability of the set of real numbers.

[13] Let us also note that the procedure of determining the S sequence proposed above is inefficient (although theoretically allowed), because due to the insolvability of the Turing machine halting problem (in the quoted Turing work we will find the appropriate proof), we do not know which of the machines generating the sequences on the list stops, and which does not (moreover: in the second case we do not know whether the machine head will not "turn" in any cycle and will not

The first exact definition of uncomputable numbers of some kind was given by the modern mathematician, Gregory Chaitin. These are Omega numbers, which for a universal Turing machine of a given type (i.e. machines with a certain number of states and symbols of the alphabet) determine the probability that a randomly selected program of operation of such a machine will stop.[14] Let us also clarify that by the program of operation is meant here the initial content of the universal machine tape, which consists of a properly coded program of the simulated machine and its initial data.[15] It refers, therefore, to the input data of the universal machine, which however strictly define its subsequent activities (the universal machine implements the program of the concrete machine for the specific data). Since the construction given by Chaitin is quite complex and serves to determine the formula for the mentioned probability (Chaitin, 1993; Chaitin, 2005), I propose here a conceptually simpler definition of another uncomputable number. I shall keep Chaitin's original idea, which refers to the issue of the halting of Turing machines.[16]

The starting point of the definition is to prepare an ordered list of programs for the universal Turing machine of certain type. As in the case of Chaitin's construction, by program I understand the initial content of the universal machine tape (including the program code of a specific machine and its input data). Since the aforementioned list is countably infinite,[17] the programs on it (with data) can be numbered as p_1, p_2, p_3 etc.

---

change the above-mentioned nth digit). We shall refer to the issue of halting further by defining an uncomputable number L.

[14] The subject literature often mentions one Omega number (see, e.g., Trzęsicki, 2006a, pp. 125–126). However, this is confusing, because for each universal Turing machine (there are infinitely many such machines) there is a separate Omega number, having a different symbolic representation.

[15] In addition to the program thus understood, each universal machine has its unique (defining it) "executive" program, which determines the way of implementation of each program placed on the tape (it regulates, among other things, how the machine head moves between the simulated machine program code and its input).

[16] Remember that this issue is expressed by the question about the existence of such a (diagnostic) Turing machine, which for each other Turing machine and each of its input data would be able to unequivocally decide whether this particular machine will stop working for this particular input or whether it will work forever.

[17] It is infinite, because due to the infinite length of the universal machine tape, there are infinitely many input data that can be put on it (despite the finite number of alphabet symbols and the finite number of states of the simulated specific machines).

Referring to this list, one can define the following binary number L in the range (0,1) : $L = 0$, $b_1b_2b_3$ ..., where bit $b_i = 1$ if the p_i program stops, and $b_i = 0$ if the p_i program does not stop (where $i \in N$).[18]

Note that the number L is strictly defined—because programs that define its subsequent bits either stop or fail. However, it is not computable, i.e. algorithmically determinable—because determining the values of subsequent bits the issue of halting cannot be algorithmically resolved in finite time. This example again shows that Turing's uncomputability is strongly associated with infinity. The number L has an infinite expansion, which is indeterminate by the finite program (indeterminate in the sense that the calculation of some of its digits would take infinitely long).

Developing the "infinity thread" in a more general context, it must be stated that all numbers that are uncomputable in the Turing sense are characterised by actual infinity (not potential[19]). Each of their symbolic representations (e.g. decimal) contains an infinite number of digits, which must be understood as an infinite whole, impossible to gradually generate, digit by digit, using any finite program (for a digital machine).[20]

---

[18] As Chaitin notes, the issue of choosing the right list, i.e. how to order the set of programs, is extremely important. It should be emphasized that it is important not only in the case of defining Omega numbers (in their case Chaitin gave a special way to specify the list), but also in the definition of another type of uncomputable numbers (Chaitin, 1993). One of the anonymous reviewers of this paper rightly stated that the type of the number L specified in the main text (computable or non-computable) depends on how the p_i program set is ordered (i.e. how the list is compiled). In particular: computable numbers (such as 2/3) can be obtained for certain orders. To solve this problem, the above-mentioned definition of Chaitin's list can be adopted. Notwithstanding the above explanations, it should be emphasized, however, that the number L is defined in such a way that even if the list in its definition causes its computability, this definition alone does not allow one to state that computability on the basis of any operations implemented by Turing machines. This is because the basis of the definition is the halting problem, and its undecidability makes it impossible to determine (in advance) which programs on the list stop and which do not. In short: perhaps for a certain list of programs the number L is computable, but we, using only the Turing machine operations, are unable to determine it.

[19] For the distinction between potential and actual infinity, see Murawski (2014). Also worth noting is the text by Witold Marciszewski on infinity (2012).

[20] The actual infinity of an uncomputable number is well illustrated by the following metaphor: if some super-algorithmic Divine Mind wanted to share with us the knowledge of an uncomputable number X, it would have to reveal it to us

Let us finish by explaining that the class of uncomputable numbers in the sense of Turing is extremely extensive, because it has the cardinality of the continuum, and therefore is equinumerous with the set of real numbers. In contrast to it, the class of computable numbers, i.e. those that are algorithmically determinable using Turing machines, has the cardinality aleph-null, i.e. is equinumerous with the set of natural numbers.[21] This disproportion between the infinities of sets of computable and uncomputable numbers seems surprising: everything that Turing machines can generate turns out to be "a drop in the ocean of uncomputability."

3. Minimum Limitations of Real Digital Codes

By real digital codes I understand here the numerical codes of the actual programs that can be physically implemented, which programs in a finite way represent functions that associate input data and results of computations. Due to the computational equivalence of (idealized) digital computers and Turing machines,[22] the results of these computations are always digital representations of some computable numbers in the Turing sense (or fragments of them, if the number has an infinite expansion).

Due to this equivalence, the general limitations of real digital codes—limitations that must be met by all programs for all digital machines—

---

in its entirety, an infinite whole, but would not be able to provide a concise algorithmic rule describing it in a finite way. This is a casual paraphrase of Chaitin's remarks (Chaitin, 1998, pp. 54–55). I write more about the difference between the types of infinity of the computable numbers (potential infinity) and the uncomputable numbers (actual infinity) in another work (Stacewicz, 2018a, pp. 180–181).

[21] This is due to the fact that all machines that generate unique strings of symbols that make up symbolic representations of computable numbers can be numbered and set into an infinite string. The set of uncomputable numbers must have the cardinality of the continuum, because it is defined as the difference of the set $\mathbb{R}$ (with the cardinality of the continuum) and the set of computable numbers (with the cardinality *aleph-null*).

[22] More precisely, each program of a certain digital machine (regardless of the technical details of its design) can be translated into the Turing machine program, in particular the universal machine program. Despite this, due to the purely physical limitations of real digital machines (not ideal, but real), not all tasks "feasible" for a UTM are feasible for them. This topic will be developed further in the main text of this section.

can be determined within the Turing model of computation, which is in the form of an abstract universal machine, called the universal Turing machine (UTM).[23] More precisely, if the solution to a certain problem cannot be coded in the form of a program for a UTM, it also cannot be considered an executable procedure for a certain digital machine.[24] Which does not mean—we must add—that it cannot be specified in the form of a procedure for a machine of another type, e.g. analogue.

From the point of view of these considerations, the key role here is played by the issue of determining uncomputable numbers, and more precisely their subsequent digits, which constitute their symbolic representations. Such numbers have correct definitions, their subsequent digits (e.g. 0 and 1) are precisely defined, and yet there is no program for the Turing machine that would allow such numbers to be determined in any finite length of time. Thus, the functions corresponding to individual uncomputable numbers—functions that bind the given accuracy (e.g., the number of the last desired digit of the decimal number expansion) to the corresponding fragment of the number—determine the limits of the digital coding. If the general solution to a given problem is reduced to this kind of function, then this solution cannot be digitally coded. To put it another way: if, for a certain problem P, each numerical code of a function that binds its input data and results corresponds to a certain uncomputable number, then this problem lies (then and only then) beyond the limits of the possibilities of digital coding. In this way, i.e. by explaining "numerically" the issue of computational unsolvability of some problems, we gain some new insight into both the reasons for, and the hypothetical possibilities of, overcoming Turing's uncomputability.

Limitations set by uncomputable numbers, and more precisely by the functions associated with them, generating their symbolic representations, should be treated as minimum limits, independent of the physical characteristics of digital machines. This statement results from the fact that the UTM machine is computationally equivalent not to physical digital machines, but to theoretical computers, with infinite memory resources and

[23] Let us remind that a universal Turing machine is a machine that, thanks to a specially selected program defining it, is able to simulate the operation of any particular Turing machine (Harel, 2000, p. 252).

[24] A wide range of uncomputable problems in the Turing model are described, for example, by Harel (2000, pp. 201–224). Gödel also mentions some important meta-mathematical problems of this type (1995/2018, p. 13).

an arbitrarily long, although finite, operating time.[25] It means that a UTM machine is able to "perform" more tasks than physical digital machines of a certain type (e.g. machines with a maximum RAM of 8 MB). Hence the conclusion that the limitations of real physical computers and the digital codes controlling them are in fact greater than the limitations of idealized machines, i.e. Turing machines. The limitations of the latter are therefore the "mathematical minimum", covering all digital computers.

Let's return to the properties of uncomputable numbers. Remember from section 2 that all representations of such numbers are characterized by infinity. These representations are in fact infinite wholes—that is, infinite sequences of symbols, which are not determined by any finite rule, having the form of a finite program for a Turing machine. From this perspective, the actual infinity of the numbers that would have to code the solutions of some problems should be considered the mathematical "cause" of Turing's uncomputability of these problems.

Due to the previously indicated correspondences between specific numbers of this type and digitally uncomputable problems (e.g. the previously determined number L corresponds to the halting problem), and the fact that the set of uncomputable numbers has the cardinality of the c o n t i n u u m , the conclusion is that uncomputable problems in the Turing sense are infinitely many, and moreover, that there are many more than there are computable ones (whose set, like the set of computable numbers, has the cardinality aleph-null). This is a conclusion, not a supposition, because each uncomputable number has at least one unsolvable problem, consisting in determining any fragment of its digital representation.

It can, of course, be argued that the infinite continuum of digitally uncomputable problems contains a relatively small number of practically relevant issues. For example, even the halting problem—as it concerns all Turing machines, and not just some of their highlighted subsets—can be considered too wide and thus insignificant from a practical point of view. However, the extremely practical point of view seems illusory. It is difficult to be sure that solutions to problems that do not translate directly into applications do not conceal practically significant consequences

[25] In the UTM model, an infinite tape is responsible for the potentially infinite memory resources and potentially infinite operating time (Stacewicz, 2018a).

(which at a given stage of the development of science and technology we do not yet know).[26]

Before moving on to the next section, devoted to alternative techniques to Turing computations, it is worth paying attention to one more feature of uncomputable numbers. In relation to the set of numbers available for Turing machines, i.e. computable ones, they are elements that, going beyond this set, allow it to be "expanded" to the form of a set of real numbers. This in turn suggests that there may be such computational techniques that refer to the theory of real numbers (and further: to some results of mathematical analysis), and, in the implementation layer, allow for operation on the physical equivalents of some or all real numbers. We'll look at the possibilities of such techniques in the next section.

4. Can There Be Effective Implementation of Non-Digital Codes?

Due to the properties of digital computers,[27] all codes representing data, programs and the results of these devices are subject to certain minimum restrictions, determined within the Turing model of computation. In fact, these restrictions consist in the inability to "go beyond" a set of computable numbers in the sense of Turing.

In connection with the above, the question arises as to whether there are any computing machines, other than digital, that would be able to operate on real, non-computable codes, i.e. certain physical representations of non-computable numbers in the Turing sense. If such machines actually existed, they could, firstly, solve problems whose only available general solutions are encoded with uncomputable numbers, and secondly, they could generate results that are such numbers (or represented by them). The computing power of such machines would, therefore, be greater than the power of digital devices.

---

[26] To justify the belief in the practical significance of any uncomputable problems, one can rely on somewhat breakneck but suggestive reasoning by analogy. Well, just as in the set of real numbers, you cannot omit (without prejudice to their mathematical utility) uncomputable numbers (because their existence gives the set R the property of continuity), so in the set of all problems you cannot miss out the set of uncomputable problems. This reasoning would require further development, which is why we only signal it in the footnote.

[27] Remember that this is about computational equivalence of (idealized) digital computers and Turing machines.

From the point of view of pure theory, such machines exist, and the general principles of their operation are determined by various models of hypercomputation—so-called because of their proper potential for expanding the capabilities of the UTM machine (Copeland, 2002). These include, among others: infinity models—allowing for an infinite number of operations (computations) in a finite time (Shagrir, 2004); non-deterministic models—describing computations initiated and/or randomly controlled (Deutsch, 1985); and analogue—allowing processing of continuous signals, mathematically described using real numbers from a specific range (Mycka & Piekarz, 2004). It is worth emphasizing that the idea of non-digital coding manifests itself most fully in the case of computations of the last type, i.e. analogue, because their theory gives the opportunity to operate on quantities (codes) from the entire continuum (and not on codes described by specific uncomputable numbers).[28]

Theoretical proposals of computations of one or another type obviously do not prejudge the issue of their physical feasibility. This issue is negatively resolved by the Church-Turing hypothesis, which in one version states that "a function is effectively computable if and only if it is computable using the universal Turing machine" (Harel, 2000, p. 240).[29] In the context of coding, this wording can be interpreted so that the only effectively processable codes are data acceptable to, and possible to generate by, the UTM machine, i.e. digital (discrete) codes. From this perspective, therefore, all codes, regardless of their theoretical description, are practically reducible to digital codes—which depends on, among other things, the fact that there is always the possibility of approximating them using digital equivalents. Considering the fact that the UTM model is theoretical and defines more computational constraints than their real possibilities, the conclusion of the hypothesis can be described in a different way. The UTM model sets absolutely minimal coding limitations in computer science.[30] In other words: all real computations—regardless of

[28] It is also worth adding that analogue techniques remain the closest to the practice of computer science—both for historical reasons (because analogue machines were already being constructed in the 1930s) and from the perspective of modern research (Mycka & Piekarz, 2004; Shannon, 1941).

[29] I treat the quoted wording as a hypothesis, because I do not prejudge whether only Turing computations (implemented in practice by digital machines) are effectively physically feasible.

[30] In the previous section, in the fourth paragraph, I also explained that these are the minimum theoretical limitations of digital techniques.

the theoretical model that describes them—must be subject to the restrictions set out in this very close to model practice (i.e. UTM). The limitations of alternative designs, e.g., analogue models, are simply broader.

The most serious arguments for the truth of the Church-Turing thesis, and therefore also for the existence of the above restrictions, refer to the concept of infinity. The basic issue is the fact that uncomputable numbers—corresponding to solutions to certain problems—are characterized by actual infinity. Remember that it concerns their endless, irregular expansions, impossible to gradually generate, which as an infinite whole represent (digitally) a given number.

The determination of such representations, and thus the resolution of the corresponding problems, must require the use of physical, uncomputable quantities existing in nature. Embedding such natural carriers of uncomputability in a machine is necessary because it is known that the overall representations of uncomputable numbers cannot be coded or determined in a traditional way, i.e. using minimally "nature engaging" binary codes and operations.[31] In particular, all effective implementations of the abovementioned analogue techniques require the use of uncomputable physical quantities. This is due to the fact that both the specificity and strength of these techniques (i.e. their greater computing power than digital techniques) rely on the possibility of processing and generating quantities from a certain *continuum* (Mycka & Piekarz, 2004). This, however, would not be continuous were it not for the uncomputable quantities filling it.[32]

Therefore, the real problem of the existence of carriers of uncomputability in nature arises. Remember that their most problematic feature is their having physical, but in accordance with the theoretical properties of

[31] Binary codes and operations must also be physically implemented using one or other natural quantities (e.g. electrical pulses); the thing is, however, that in their case it is enough to use any physical quantities that are easily distinguishable (or even one recognizable quantity and the lack thereof). Thus, the degree of "engagement" of nature is minimal in their case.

[32] The same fact can be expressed by referring to the properties of real numbers, which are the mathematical equivalent of processed continuous analogue signals. Well, without uncomputable numbers, each range of real numbers (equivalent to the physical domain of analogue signals) has the cardinality *aleph-null*, so it is equivalent to a discrete set of natural numbers.

uncomputable numbers, actual infinity.[33] If such carriers existed, the set of practical computational codes would go beyond the set of digital codes. It would include codes that have direct roots in nature. Some of their components, at least, would simply be "calls" to natural phenomena that would return some uncomputable quantities directly and in whole. In particular, the theory of analogue-continuous computation initiated by Claude Shannon (Shannon, 1941) states that a complex analogue code may include elementary integration operations, whose continuous results (implemented in real time) must be obtained by measuring phenomena occurring in special physical systems (e.g. electronic integrators). And as I wrote above, for the continuity of the result set it is necessary for it to contain uncomputable quantities.

The existence of *uncomputable* natural phenomena—that is, those that cannot be described in terms of computable numbers and functions implemented by Turing machines—postulates certain physical theories. One particularly cited example is from Pour-El and Richards (1989). According to it, the three-dimensional wave described by a certain differential equation can obtain that can be expressed only by means of uncomputable numbers. John Doyle's proposals which indicate the inability to describe the processes of achieving equilibrium occurring in nature (e.g. thermodynamic) using computable functions fall into the same category (Copeland, 2002, p. 470). These and other examples seem to indicate the real existence of phenomena that we could treat as natural carriers of uncomputability. Let us remember, however, that empirical tests are responsible for the compatibility of physical theories with reality, which no finite number (again, an infinity problem!) can ever confirm with 100% certainty.

Suppose, however, regardless of the above objection of an epistemological nature, that physical carriers of uncomputable codes exist and can be used as part of one or other natural computations.[34] Despite this assump-

[33] The philosophical argument for the existence of infinite quantities in nature is contained in *Amor Infiniti. What philosophical intuitions lead to it?* (Marciszewski, 2012).

[34] I am thinking of computation designed by man, but involving significantly substrates and/or natural processes (e.g. quantum calculations or those performed using DNA molecules). The class of natural computation in a broader sense also includes: 1) computation inspired by observation of nature (e.g. implemented by artificial neural networks) and 2) processes occurring in nature, described in com-

tion, another problem arises concerning the possibility of *reading*, and thus knowing the obtained result. The problem is that in order to know the result, infinite accuracy in reading the entire uncomputable quantity is necessary.[35] It is necessary, because finite accuracy, which, after all, characterizes all real measuring instruments, would bring the uncomputable number desired in a given situation to the level of a finite computable quantity. Therefore, we would lose the expected effect of overcoming the limitations of digital computing. It can be argued that for some problems, it is enough for uncomputable quantities to be simply processed and not read—because the solution to the problem is some specific finite value that can be read (Stannett, 2003, pp. 121–123). The approach considered here is, however, about knowing the general solution to the problem (a function that associates all possible input data with the corresponding results), and this type of solution is encoded by an entire number that is uncomputable with actually infinite expansion. Therefore, the epistemic problem remains: without infinite accuracy of reading we cannot know such a solution.

To conclude: the actual infinity of uncomputable numbers means that the limitations of computational techniques suggested by the Church-Turing thesis—techniques that require the physical implementation of certain computational codes—can be overcome under at least two conditions: 1) the occurrence of infinite quantities in nature that can be recorded and processed, 2) the existence of a mental disposition for insight into actually infinite objects and their relations and methods (e.g. methods of defining). The second condition must be considered fulfilled—as evidenced by the actual infinity theories created by people, including theoretical models of computing on actually infinite quantities. The possibility of meeting the first condition seems, at least, problematic.

putational categories (e.g. intracerebral processes; see Kari & Rozenberg, 2008; Rozenberg, Back, & Kok, 2012).

[35] Such accuracy is necessary in the case of analogue techniques, which by definition operate on continuous quantities (two quantities in the continuous domain may differ from each other by any small amount).

REFERENCES

Angius, N., Turner, R. (2017). Philosophy of Computer Science. *Stanford Encyclopedia of Philosophy*. Retrieved from: https://plato.stanford.edu/entries/computer-science/

Chaitin, G. J. (1993). Randomness in Arithmetic and the Decline and Fall of Reductionism in Pure Mathematics. *Bulletin of the European Association for Theoretical Computer Science*, *50*, 314–328.

Chaitin, G. J. (1998). *The Limits of Mathematics*. Singapore: Springer.

Chaitin, G. J. (2005). Omega and Why Maths Has No TOEs. Retrieved from: https://plus.maths.org/content/os/issue37/features/omega

Colburn, T. R. (2000). *Philosophy and Computer Science*. Armonk, NY: M.E. Sharpe.

Copeland, J. (2002). Hypercomputation. *Mind and Machines*, *12*(4), 461–502.

Deutsch, D. (1985). Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of The Royal Society of London A*, *400*, 97–117.

Etesi, G., Nemeti, I. (2002). Non-Turing Computations via Malament-Hogarth Space-Times. *International Journal of Theoretic Physics*, *41*(2), 341–370.

Gödel, K. (1995/2018). O pewnych zasadniczych twierdzeniach dotyczących podstaw matematyki i wnioskach z nich płynących. *Studia Semiotyczne*, *32*(2), 9–32.

Harel, D. (2000). Rzecz o istocie informatyki. *Algorytmika*. Warsaw: Wydawnictwa Naukowo-Techniczne.

Kari, L., Rozenberg, G. (2008). The Many Facets of Natural Computing, *Communications of the ACM*, *51*(10), 72–83.

Krajewski, S. (2014). Neopitagoreizm współczesny: uwagi o żywotności pitagoreizmu. In: M. Heller, S. Krajewski (Eds.), *Czy fizyka i matematyka to nauki humanistyczne?* (pp. 348–366). Kraków: Copernicus Center Press.

Leibniz, G. W. (1890). *Philosophische Schriften* (Vol. VII). Berlin: Weidmann.

Marciszewski, W. (2012). Amor Infiniti. Jakie doń prowadzą intuicje filozoficzne? Retrieved from: http://marciszewski.eu/?p=2955

Marciszewski, W., Stacewicz, P. (2011). *Umysł-Komputer-Świat. O zagadce umysłu z informatycznego punktu widzenia.* Warsaw: Akademicka Oficyna Wydawnicza EXIT.

Moor, J. H. (1978). Three Myths of Computer Science, *The British Journal for the Philosophy of Science*, *29*(3), 213–222.

Murawski, R. (2014). Nieskończoność w matematyce. Zmagania z potrzebnym, acz kłopotliwym pojęciem. *Zagadnienia Filozoficzne w Nauce*, *55*(2), 5–42.

Mycka, J. M., Piekarz, M. (2004). Przegląd zagadnień obliczalności analogowej. In: S. Grzegórski, M. Miłosz, P. Muryjas (Eds.), *Algorytmy, metody i programy naukowe* (pp. 125–132). Lublin: Polskie Towarzystwo Informatyczne.

Mycka, J. M., Olszewski A. (2015). Czy teza Churcha ma jeszcze jakieś znaczenie dla informatyki? In: P. Stacewicz (Ed.), *Informatyka a filozofia. Od informatyki i jej zastosowań do światopoglądu informatycznego* (pp. 53–74). Warsaw: Oficyna Wydawnicza Politechniki Warszawskiej.

Ord, T. (2002). Hypercomputation: Computing More Than the Turing Machine. Retrieved from: https://arxiv.org/ftp/math/papers/0209/0209332.pdf

Ord, T. (2006). The many forms of hypercomputation. *Applied Mathematics and Computation*, *178*(1), 8–24.

Pour-El, M. B., Richards, J. I. (1989). *Computability in Analysis and Physics*. Berlin: Springer.

Rozenberg, G., Back, T., Kok, J. N. (2012). *Handbook of Natural Computing*. Berlin-Heidelberg: Springer.

Rubel, L. (1993). The Extended Analog Computer. *Advances in Applied Mathematics*, *14*(1), 39–50.

Shagrir, O. (2004). Super-Tasks, Accelerating Turing Machines and Uncomputability. *Theoretical Computer Science*, *317*(1–3), 105–114.

Shannon, C. (1941). Mathematical Theory of the Differential Analyzer. *Journal of Mathematics and Physics*. *20*(1–4), 337–354.

Stacewicz, P. (2012). Co łączy umysł z teorią liczb? *Filozofia Nauki*, *79*(3), 111–126.

Stacewicz. P. (2015). Informatyczne kłopoty z nieskończonością. In: R. Murawski (Ed.), *Filozofia matematyki i informatyki* (pp. 310–327). Kraków: Copernicus Center Press.

Stacewicz, P. (2018a). Czy informatykom musi wystarczyć nieskończoność potencjalna? In: R. Murawski, J. Woleński (Eds.), *Problemy filozofii matematyki i informatyki* (pp. 177–190). Poznań: Wydawnictwo Naukowe Uniwersytetu im. Adama Mickiewicza w Poznaniu.

Stacewicz, P. (2018b). O teoretycznej (nie)zbędności kategorii liczby w informatyce i jej metodologii. Retrieved from: http://marciszewski.eu/?p=999

Stannett, M. (2003). Computation and Hypercomputation. *Minds and Machines*, *13*(1), 115–153.

Trzęsicki, K. (2006). From the Idea of Decidability to the Number Omega. *Studies in Logic, Grammar and Rhetoric*, *22*(1), 73–142.

Trzęsicki, K. (2006). Leibnizjańskie inspiracje informatyki. *Filozofia Nauki*, *55*(3), 21–48.

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *2– 42*(1), 230–265.

Originally published as "Liczby nieobliczalne a granice kodowania w informatyce". *Studia Semiotyczne*, *32*(2), 131–152, DOI: 10.26333/sts.xxxii2.08. Translated by Martin Hinton.