Jerzy Mycka

Maria Curie-Skłodowska University
Institute of Mathematics
e-mail: jerzy.mycka@umcs.lublin.pl
ORCID: 0000-0002-4029-3122

Adam Olszewski

The Pontifical University of John Paul II in Cracow
Philosophy Department
& Copernicus Center for Interdisciplinary Studies
e-mail: adam.olszewski@upjp2.edu.pl
ORCID: 0000-0003-3069-7518

IS CHURCH'S THESIS STILL RELEVANT?*

Abstract. The article analyses the role of Church's Thesis (hereinafter CT) in the context of the development of hypercomputation research. The text begins by presenting various views on the essence of computer science and the limitations of its methods. Then CT and its importance in determining the limits of methods used by computer science is presented. Basing on the above explanations, the work goes on to characterize various proposals of hypercomputation showing their relative power in relation to the arithmetic hierarchy.

The general theme of the article is the analysis of mutual relations between the content of CT and the theories of hypercomputation. In the main part of the paper the arguments for abolition of CT caused by the introduction of hypercomputable methods in computer science are presented and critique of these views is presented. The role of the efficiency condition contained in the formulation of CT is stressed. The discussion ends with a summary defending the current status of Church's thesis within the framework of philosophy and computer science as an important point of reference for determining what the notion of effective calculability really is. The considerations included in this article seem to be quite up-to-date relative to the current state of affairs in computer science.¹

Keywords: hypercomputation, Church's thesis, effective computations, computer science & philosophy.

1. What is computer science?

The title question of the article is a reference to the title of Mark Burgin's article *The Rise and Fall of Church-Turing Thesis*, in which the author questions the value of Church's thesis for computer science at the present (and future) stage of its development.² As we know, almost eighty years after the formulation of CT, its logical value has not been resolved, which

is surprising in itself. The traditional view with regard to CT is that it sets an upper limit to a class of functions effectively calculable³ in an intuitive sense, equating it (this class) with a class of (partially) recursive functions. So, if we consider an algorithm to be the basic tool of computing, then CT severely limits the framework of what a computer scientist (IT specialist) can get, especially in terms of practical use. The task of this work is to defend the unique role of CT for computer science in the context of the whole class of proposed models of calculations having the feature of hypercomputability (so-called hypercomputations).⁴ In the first part we will briefly describe the different concepts of computer science and computing, in the second part we will consider more philosophical and general issues related to hypercomputability and to the rejection of CT, and in the third part we will present a more detailed critique of Burgin's ideas.

Let us start by trying to define what computer science is.⁵ There is probably no single definition on which there would be general agreement. And so, Newell and Simon point out that computer science is an empirical science, where every program (and every computer too) is an experiment (Murawski (2014), p. 9 ff.). The authors of this view talk about 'listening' to the answers from a machine previously constructed. This indicates that programmers and engineers are the creators of programs and machines, some of their properties are not known to the creators *a priori* but can only be discovered by *a posteriori* observation. This shows that the word 'experiment' is used here in a rather special sense. Donald Knuth thinks that computer science is concerned with studying algorithms. Algorithms are understood as precisely defined sequences of rules giving instructions how to get a certain output from a given input in a finite number of steps. The program for a given algorithm is its concrete realization, and a similar relationship exists between information and computer data. With such an approach, computer science has existed for many years (for example, the Babylonians were already computer scientists 3500 years ago). Knuth also recalls Forsythe's view that an algorithmic viewpoint is a useful way of organizing knowledge in general. Stuart C. Shapiro treats computer science as a natural science that studies procedures (Murawski (2014), p. 21), with procedures being understood very broadly as a particular, concrete way to achieve something. Procedures in computer science do not have to be organized step-by-step, because parallel procedures are allowed, but they cannot be endless or inefficient (such as heuristic procedures). According to Shapiro, computer science is a natural science, because by studying procedures it studies phenomena occurring in nature. Computer science understood in this way is divided into theoretical and applied parts. Frederick P. Brooks Jr. de-

scribes computer science as a synthetic and engineering discipline, which is not a science, and which produces things, whether they are computers or algorithms or programs (cf. Murawski (2014), p. 10). Brooks draws attention to the publications of computer scientists, which, like in mathematics, are increasingly detached from real problems and from the possibility of being understood by readers and even some specialists, making them 'junk'. Peter J. Denning understands computer science as the science of techniques (such as design, analysis, implementation, efficiency and application) of information processing. According to him, the individual feature of computer science is that it is impossible to precisely distinguish the theoretical part from the engineering part (applications), which in the case of other sciences is usually possible. It is worth quoting a description of all stages of the evolution of computer science according to Denning, which are (from the earliest to the newest): theory of computation (1940), numerical calculations (1945), architecture (1950), programming languages and methodology (1960), algorithms and data structures (1968), operating systems (1971), networks (1975), interface software (1978), database systems (1980), parallel calculations (1982) and artificial intelligence (1986?) (cf. Murawski (2014), p. 31 ff.). Juris Hartmanis and Herbert Lin see computer science as a science focused on how information is represented and processed and analysing the devices and systems that perform these tasks. The central concepts for the whole computer science are algorithmic thinking, information representation and computer programs. These authors provide a nice metaphor of this view saying that programmers are to a large extent playwrights and puppet actors (Murawski (2014), p. 35).

Summarizing these different descriptions of computer science, it can be seen that there is no general consensus on the scientific character of this discipline (in the sense of natural or formal science), although most authors share this view; secondly, there are usually two parallel components – theory and engineering; thirdly, there is often an empirical element; and fourthly, the basic concepts are algorithms (procedures), programs and machines (computers) that process information (which involves the concept of structured data). The authors of this article agree that computer science is most conveniently understood as a discipline that has two components: theoretical and practical. The first component – theoretical – deals with algorithms and programs *in abstracto*, i.e. construction of programs and examination of their general properties, while the second component deals with implementations of a specific computation on appropriately constructed machines (computers).⁶ CT in this context could be seen as a kind of a link (or a bridge) between these two components.

2. What is hypercomputation?

Stuart C. Shapiro, in his brief article *Computer Science: The Study of Procedures* (discussed above, see (Murawski, 2014)) wrote that computer science is based on some fundamental principles. One of them is Church-Turing's thesis, which implies that computers can perform any procedure performed by any machine (in the broad sense of the term, including animals or even humans as long as their actions are mechanistic). Referring to the idea of Hilbert, who demanded that all scientific disciplines should be axiomatized, one can ask about the possibility of axiomatization of computer science. Giving axioms to computer science would be very helpful in understanding it more precisely. It seems that CT would be among the axioms of this discipline.⁷ The CT 'identifies', as Alonzo Church wanted, an intuitive concept of an effectively calculable function with a mathematical concept of recursive function, or a Turing machine.⁸ This 'identity'⁹ allows the application of a meta-rule for extensions of concepts which can be given as [MEP]: for any concepts $P1$, $P2$ and the context C occurs: if $P1(id)P2$ and $C(P1)$, then $C(P2)$. It can be interpreted in the following way: if $P1$ inserted in some context C (i.e. C is a sentence or a group of sentences using the notion $P1$) gives truthful statement, then after substituting $P2$ for $P1$ in C we will still get the truth. This rule is debatable, but if we adopt it with CT, we will get quite interesting conclusions.¹⁰

One of them concerns the unsolvability of the halting problem. This proposition and its proof were formulated by Alan Turing in the case of Turing machines. It states that there is no Turing machine that solves the problem whether the Turing machine specified by its program will stop for a given input. The proof for this assertion is not straightforward, it starts by assuming that the machine that solves the problem exists and then shows that this assumption leads to a contradiction. Under CT and MEP, this statement is equivalent to the statement that there is no function which is intuitively computable and able to solve the halting problem for Turing machines. The truth of this assertion is not so obvious, as the class of intuitively computable functions can be larger than the Turing-computable functions. Turing's result of the unsolvability of the halting problem is one of the assertions about the impossibility of achieving something by (natural reasoning of) a man as a species – homo sapiens. This has triggered a variety of reactions not only among scholars, but also among ordinary people. Davis (cf. Davis (2004, 2006)) compares this situation to the disagreement of some people with the impossibility of constructing *perpetuum mobile*. Whole

crowds of constructors (including scientists) tried, despite a clear argument about the impossibility, to show that *perpetuum mobile* can nevertheless be constructed. Ord (Ord, 2006), on the other hand, cites as an analogy the example of the emergence of non-Euclidean geometries, which were initially even rejected as having nothing to do with physical reality. Finally, it turned out that the global structure of the universe is probably given by a non-Euclidean geometry. It seems that the starting point of such hopes is to negate CT, and consequently to reject the MEP. The above metaphors of Davis and Ord are quite attractive, but if we look at them closely, they turn out to be not quite accurate. Non-Euclidean geometries are not extensions of Euclidean geometry, but they are some alternatives to it¹¹ – unlike hypercomputation, which is (at least most often) some extension of standard calculations. On the other hand, *perpetuum mobile* is impossible to construct as it is incompatible with the principles of thermodynamics and energy conservation, i.e. with the basic principles of classical physics. However, it can be suggested that just like the second law of thermodynamics – in some sense – defines the boundaries within which reasonable physical theories must fit, CT can be seen as a similar kind of constraint imposed on realistic theories of computability.¹² Hypercomputation – at least according to some concepts – is not excluded, but is even acceptable from a physical point of view.¹³ The classic example of such argumentation is the work of Shagrir and Pitowsky (cf. Shagrir and Pitowsky (2003), Zenil (2012)), where a description of a physical digital hypercomputer based on the principles of general theory of relativity and performing an infinite number of operations in a finite period of time is given.

The first attempt to go beyond the limits of functions computable in the sense of Turing machines (and on the basis of CT also beyond functions effectively computable in the intuitive sense) was the concept of oracle, which Turing himself presented in his doctoral thesis written under the supervision of A. Church and completed in 1939. Turing machine (TM) equipped with an oracle (for some function or some set) – oTM – allows for obtaining answers based on a computation but also depending on responses of the oracle. Interesting cases are when the oracle is not recursive function: then we have a kind of a black box, which can extend possible calculations made by Turing machines. This concept has given rise to a study of so-called relative computability, which is part of a generalised theory of computation that analyses the computing power of TM taking into account the capabilities of the attached oracle.¹⁴ Although Turing did not deal with this issue himself, the proponents of hypercomputations continue this topic, considering the nature of the oracle itself: it can be a black box with an undisclosed

structure, or an infinite tape with a record of the characteristic function of some infinite set, or an input stream providing nonrecursive data from various sources, mainly from nature (e.g. quantum mechanics or other physical processes).¹⁵ In some situations, such an oTM may calculate functions that are uncomputable by ordinary TM (see Ord (2006), p. 146).

Another strategy of hypercomputable innovations used in the analysis of capabilities of Turing is based on modifications of their different components: the number of states (two states can replace any finite number of them); the number of symbols (two are sufficient to replace any finite alphabet); operations (the operation of erasing symbols can be abandoned without losing any computational power); the number of tapes and heads (each finite system is computationally equivalent to one tape with one head); nondeterminism (abandoning the machine's determinism does not increase its power).¹⁶ As it was demonstrated, these attempts did not actually bring anything significant, as they did not increase the computational power of Turing machines (as opposed to some types of oracles).

The main attack on CT came from the emergence of a notion of the so-called hypercomputations. The common feature of this approach is in fact the weakening of the concept of effective calculability, resulting in a new (intuitive) concept of computability under which fall more functions. In this case CT becomes false and, what is more, loses its special character. Hypercomputations have different forms, but the same general objective: to add some new methods of computation to solve problems not available within the limits of classical computability.

Their rationale is connected to, among other things, three main interpretations of CT (derived from ideas of G. Kreisel), which differ in their understanding of effective computability:

1. human variant (a function effectively computable is a function calculated by [the mind] of a man),
2. physical variant (a function effectively computable is a function that is realized [relative to recognized scientific theories] by some natural process),
3. machine variant (a function effectively computable is a function that is calculated mechanically).

A separate thesis is that the three variants are equivalent to each other. We will give below a few examples of some variants or versions of hypercomputations.

One version of hypercomputation is created with the concept of randomness (and usually on the physical level by use of quantum mechanics). The idea of the so-called quantum computer was created by consideration

of some quantum system, the evolution of which is understood as a calculation. Data is coded as the stream of cubits (quantum unit of information), which are represented by elementary particles. However, it turned out that the superiority of quantum computers over ordinary machines can in fact rely on much higher computing speed. In the light of the research carried out so far everything indicates that quantum machines bring a significant change only at the level of computational complexity (the BPQ class of problems solved by quantum machines in polynomial time is a superclass of the class of problems solved in polynomial time by ordinary Turing machines; although it is suspected to be a proper superclass, so far no proof of this hypothesis has been given).

Hypercomputation could also arise if there is a certain process in nature which is not recursive (not computable in a classical sense) and which, in order to be useful, would have to meet the condition of being harnessable, i.e. it would have to be a 'stream that has the information properly encoded in itself.' Harnessability (harnessing) of the process can be understood in the two ways indicated below (cf. Ord (2002), p. 16):

Some process P is mathematically harnessable if and only if behaviour of P with regard to its input/output:

- is either directly deterministic or deterministic in an approximate manner (i.e., one that guarantees that the chance of an error and a size of this error can be reduced accordingly to any given restriction),
- has a finite mathematical definition that can be ascribed to this process.

Process P is physically harnessable if and only if it can be demonstrated that a process is useful as a simulation of some kind of computations. For example, some possibility to calculate nonrecursive functions would be given by physical processes growing faster than the Busy Beaver function. However, the existence of such processes is uncertain (see Ord (2006), p. 146).

Another possibility of hypercomputation is the fair-nondeterminism, associated with non-deterministic TM. Ordinary non-deterministic machines are equivalent (in their computational power – not in their performance) to standard TMs, and their nondeterminism is called bounded nondeterminism. Some insights from physics (Hewitt), client-cloud computing and concurrent processing¹⁷ have led to the concept of unbounded nondeterminism. One of the main problems of concurrent processes is the problem of communication between processors. This concept is related to fairness (impartiality). The meaning of this demand is that if a machine adopts some state infinitely many times, it must make every transition from that state to any state allowed in the program (see Spaan et al. (1989), pp. 3-4). An example of a program that distinguishes between results of bounded

nondeterministic machines and unbounded nondeterministic machines (satisfying fairness condition) can be described as follows:

1. write the sign 1 in the next cell and go to 2 or jump to 3,
2. go to 1,
3. stop.

This machine can work indefinitely, but if it has fairness property, it will always stop, although we do not know when. Any nondeterministic system of this type meets the following conditions: 1) when it starts, the system will always stop; and 2) for any natural number n , it is possible that the system will stop at an output which is greater than n (cf. Spaan et al. (1989), p. 11). The term unbounded nondeterminism means that the calculation by such machines will always end, but the number of steps to stop is unknown and unbounded. Machines that would meet the conditions described above could solve the halting problem. The design of such a non-deterministic Turing machine H to solve this problem for any machine M and its data d can be sketched as follows. For the given code of some Turing machine M and its data d , the non-deterministic machine H shall make in the state q one of the following choices:

- write 1 in the next cell and return to the state q ,
- or use the number n of all written signs of 1 to simulate n steps of M on data d ; if in the course of this simulation machine M is in its final state then H goes into the accepting state, otherwise (if n steps are not enough for M to calculate the result for data d) machine H goes into the rejecting state.

The condition of fairness ensures that the second choice must always appear during the calculation of H (although there is no indication to when it would appear). Every possible calculation path H is finite; if machine M stops at all for data d , one of the calculation paths H ends with acceptance. This meets the standard acceptance condition for non-deterministic machines – at least one path gives acceptance; all possible calculation paths are finite.

The considered extensions of the concept of computability are directly or indirectly related to the concept of infinity. Strictly speaking, infinity already appears with the standard TM concept, since the machine under consideration may not stop for a certain input, which means that it will work infinitely. However, to obtain hypercomputation we can try alternative use of infinity. One way is to accept infinite input. The classic TM allows only finite input, but a natural extension could accept an appropriately coded real number on the input tape. TM can produce the output of any length, provided that it gets an infinite representation of some real number at the input. It can be assumed that this is some form of hyper-

computation, since the calculation is done on real numbers. This idea is used in recursive analysis.¹⁸ In practice, the calculation is done on a finite part of a number, that is to say, *de facto* on finite numbers and, apart from a convenient notation in some cases, does not expand the computing power of TMs.¹⁹

Another way of attaching infinity to ordinary TM is to allow an infinite number of internal states (cf. Ord (2002)). Such machines have an infinite program that assigns to each state some procedure describing a computable function and is a little bit like *o*-machines. In this case, the machine may calculate any function from a set of natural numbers into a set of natural numbers. But if we do not impose some restrictions on the organization of this infinite set of states then in the case where the rest of the (model) structure is compatible with the classical Turing machine, we will lose all efficiency.

Turing, in his analysis of computability, did not assume anything special about the time needed to perform particular operations. He believed that time is of a discrete nature, and for each calculation step such a discrete (equal) unit of time is needed. This 'gap' in Turing's argumentation became an opportunity to introduce the so-called accelerating Turing machines (ATMs), which perform each subsequent calculation step in time twice as short as the previous one (cf. Shagrir (2004)). In this way such machines could perform an infinite number of operations in two units of time (e.g. in two minutes). As a result, we can construct the ATM machine which can simulate any given TM. It is easy to see that this procedure solves the halting problem. Quite naturally, we can also think of further generalizations of TM that will perform an infinite number of operations, using ordinal numbers as their measure. These infinite time TM (ITTM) perform an infinite number of operations using some ATM, then resets the ATM, activating it again on the output data obtained in the previous calculation (modifying the time proportions accordingly). This procedure can be repeated an infinite number of times. Such Turing machines could carry out calculations using some ordinal number of steps (cf. Ord (2002), p. 21).

An interesting attempt in looking for a hypercomputation based on a physical process was developed by Mark Hogarth, who speaks of the possibility – in the sense of being consistent with general theory of relativity – of some special process that uses and 'tames' infinity. The process itself is infinite according to its internal time, while for the observer it lasts a finite time. A standard TM could, according to this idea, make an infinite journey along some line of the world to solve the halting problem and could pass the result to an external observer.²⁰

The recurrent neural network is another attempt to introduce the hypercomputation. If the weights assigned to connections between vertices (nodes) are rational numbers, the network's capabilities do not exceed standard TM. However, assigning weights that are real numbers significantly increases the computing capabilities of such a network.²¹

It has also been proposed that hypercomputation can be based on new types the domain and counterdomains of algorithms: e.g. real numbers or elements of any ring. In such an approach, under certain assumptions, it is possible to obtain 'computability' of classically uncomputable functions.²²

In the paper (Ord, 2006) we find an interesting comparison of the main theories of hypercomputations in relation to their position in relation to the arithmetic hierarchy, which allows us to compare these theories among ourselves. Below we present this list of models.

Table 1
Computability according to the place within (or outside)
the arithmetical hierarchy

Turing machine	Δ_1
accelerating Turing machine	Σ_1
infinite Turing machine ($\omega \times n$ steps)	Δ_n
infinite Turing machine ($\omega \times \omega$ steps)	Δ_ω
infinite Turing machine	$\Sigma_1 \cup \Pi_1 < \dots < \Delta_2^1$
Malament-Hoghart machine	up to Δ_ω
fair non-deterministic machine	Σ_1
Turing machine with infinite number of states	<i>all levels</i>
periodic neural network (exponential time)	<i>all levels</i>
periodic neural network (polynomial time)	<i>P/poly</i>
oracle machine (o-machines)	<i>all levels</i>

The above table uses the levels of the arithmetic hierarchy to classify the computing power of different hypercomputation models. This is the hierarchy that can be considered as some kind of a scale for non-computable processes. The arithmetic hierarchy is usually defined as two sequences of families of relations on natural numbers. It is a fairly natural choice, given that relations can represent both sets and functions.

The starting point is the family of such relations R , that their satisfiability for a given sequence of arguments $(n_1, \dots, n_k)$ can be checked by a (standard) Turing M_R machine. Such an M_R machine for the tape representation

of numbers $(n_1, \dots, n_k)$, performs the appropriate calculations and ends in the accepting state when $R(n_1, \dots, n_k)$ holds; and in the rejecting state when $R(n_1, \dots, n_k)$ does not hold. This family of relations establishes the level Σ_0 (or equivalently the level Π_0). At subsequent levels, relations from lower levels are modified by adding quantifiers: the relation $R(n_1, \dots, n_k)$ is at Σ_{n+1} if and only if $R(n_1, \dots, n_k) \leftrightarrow \exists n_{k+1} Q(n_1, \dots, n_k, n_{k+1})$ for Q belonging to Π_n ; symmetrically, the relation $R(n_1, \dots, n_k)$ is at Π_{n+1} if and only if $R(n_1, \dots, n_k) \leftrightarrow \forall n_{k+1} Q(n_1, \dots, n_k, n_{k+1})$ for Q belonging to Σ_n .

Intuitively, this means that to decide whether the relation R is at the level of Σ_1 , it is necessary to check with the Turing machine that the relation Q is satisfied for consecutive values of n_{k+1} . The acceptance of R will occur when the first number satisfying Q is found. However, the rejection of R can only be announced if Q is negatively verified for all natural numbers. Similarly, to check the relation R from the Π_1 level, a witness must be found for not satisfying Q or all natural numbers must be positively verified as satisfying Q . At higher levels, further quantifiers appear and the superimposition of them intensifies the use of an infinite number of steps: for Σ_n and Π_n the process of working in an infinite number of steps is repeated n times. The common part of Σ_n and Π_n is denoted by Δ_n and the sum of all relations families in the arithmetic hierarchy by Δ_ω .

As we can see, the higher a certain relation is in the arithmetic hierarchy, the greater is the number of infinite processes that are necessary to check it for given arguments. What is important is that the arithmetic hierarchy is related to Turing's machines with oracles. Let us remind that $\emptyset^{(n)}$ is a set constructed as a result of n -times application of a jump operation – i.e., a particular kind of a construction which encapsulates the solution of some instance of the halting problem. It turns out that the relation is at the level of Δ_{n+1} if and only if it can be verified by a machine with an oracle $\emptyset^{(n)}$. As we can see, the oracle for the halting problem gives Turing machine a real gain in computing power, by which it has some new capabilities for infinite calculations.

In those places of the table where the symbols Σ_1^1 , Π_1^1 , Δ_2^1 appear, it means some families from the analytical hierarchy. In this case, in relation definitions, quantifiers with variables representing sets are allowed (unlike in the case of the arithmetic hierarchy, where quantifiers only refer to numbers). The whole arithmetic hierarchy is contained in the basic level of the analytical hierarchy, and the degree of entanglement of infinite processes with relations contained in subsequent levels of the analytical hierarchy is proportionally much greater.

The *P/poly* class should also be explained. It is defined as a class of problems that can be solved by an ordinary Turing machine of polynomial time complexity, which can use a sequence of clues of polynomial size depending on data size. What is crucial, however, is that the clue string is heterogeneous – that is, there is no computationally unified method of determining it. In other words, an infinite resource appears again as an essential component of hypercomputation.

3. What is possible and realistic in computer science?

Mark Burgin, who himself has created several alternative concepts to expand the concept of computability, is absolutely convinced of the need to go beyond the model of computability created by Turing (and equivalent models). He enumerates the following directions in which hypercomputation is developing:²³

A. *Algorithmic models*:

1. *Limiting recursive functions* (Gold, 1965; Putnam, 1965)
2. *Inductive inference* (Gold, 1967; Blum and Blum, 1975)
3. *Inductive centralized computations* (Freuvald, 1974; Burgin, 1983; Burgin, 1987; Burgin, 1999)
4. *Inductive net computations* (Garson and Franklin, 1989; Garson, Franklin, Bagget, Boyd and Dickerson, 1992)
5. *Topological computations* (Burgin, 1983; 1992; Pour-El and Richards, 1989)
6. *Fuzzy computations* (Zadeh, 1969; Wiedermann, 2000)
7. *Interactive and concurrent computations* (Milner, 1973; 1980; Milne, 1985; Hoare, 1985; Wegner, 1995)

B. *Semi-algorithmic models*:²⁴ Computations with an oracle (Turing, 1939)

1. *Infinite precision net computations* (Pollack, 1987; Hartley and Szu, 1987; Siegelman and Sontag, 1991)
2. *Recursion theory on real numbers* (Abramson, 1971; Blum, Shub and Smale, 1989; Moore, 1996)
3. *Analogue computations* (Shannon, 1941; Scarpellini, 1963; Moore, 1996)
4. *Infinite time Turing machines* (Hamkins and Lewis, 2000; Welch, 2000)
5. *Trans-recursive operators* (Burgin and Borodyanskii, 1992)
6. *Dynamical system and field computations* (Mesarovic and Takahara, 1975; MacLennan, 1990; da Costa and Doria, 1996; Bournez, 1999)

7. *Fixed point models* (Scott, 1971; Manna and Vuillemin, 1972; Abramsky and Jung, 1994; Edalat, 1997; Edalat and Sünderhauf, 1998)

C. *Abstract models:*

1. *α -recursion or recursion on ordinals* (Takeuti, 1960; Machover, 1961; Levy, 1963; Wainer)
2. *Generalized computability* (Moschovakis, 1969; Skordev, 1976; Tucker and Zucker, 1988; 2002)
3. *Recursion in higher types* (Kleene, 1959; Kleene, 1963)
4. *Induction in abstract structures* (Moschovakis, 1974)
5. *Computations in categories* (Riguet, 1965; Adamék, 1974; Arbib and Manes, 1974; Trnková, 1974)

The hypercomputation models indicated above can be ordered on the basis of various criteria. On the basis of the analysis of various models of hypercomputable processes from the above presented list (among which a special role is played by inductive Turing machines) Burgin (cf. Burgin (2002)) concludes that CT can be considered overturned.

However, it seems that Burgin's arguments do not give reasons for such bold statements. Burgin strongly emphasizes the new perspectives created by hypercomputation and the need for computer science practice to catch up with the new theories of computations. This way of thinking seems to completely overlook one of the important aspects of the CT: the determination of the relation between the practical and theoretical possibilities of computational functions. There is no reason to believe uncritically that every constructive – in some mathematical sense – theory can be considered as fully applicable in practice. Therefore, it is important to ask which of the mathematical concepts correspond to practically useful mechanisms. CT gives a certain answer to this question and this answer is rooted in many years of practice and theoretical reflection. Therefore, alternative proposals should not be based on pure declarations and uncritical optimism but should have verifiable justifications.

Mark Burgin presents the following argumentation criticising the existing models of computation. Until recently, the dominant models were based on closed systems (without the possibility of interaction), working in a sequential manner in a finite time. Today's reality shows us computer applications running cyclically, interacting with the environment and using parallelism mechanisms. Therefore, according to Burgin, a new theory should be introduced, adapted to the modified practice, using the theory of hypercomputation and applying it to new reality.

The problem with this kind of argument is that it ignores the difference between:

1. incompatibility of some details of theoretical models with practically constructed computer systems and
2. relationship of computational power of these models and their counterparts – i.e. real world computers.

It is true that an ordinary Turing machine is not a good model of modern computers. As a result, many other models of computational processes are proposed. However, this does not change the fact that so far no one has been able to indicate a device which computing power would exceed the barriers built into the Turing machine. Thus, although the shape of the models is sometimes inadequate for modern computing machines, their computational power is equivalent.

Burgin tries to overcome these objections by pointing out that hypercomputable processes go beyond the schemes of classical algorithmics. He mentions the following factors as elements that introduce this new quality:

1. interactions of computational processes,
2. possibilities of infinite computations.

We will consider these two points from both theory and practice point of view, remembering our observations made while discussing various models of hypercomputation in the previous parts of this article.

Interaction is not a completely new discovery of computer science theory (cf. Hewitt (2013)). Turing machines with oracles can be considered as a model of additional information acquisition from an external source during their computations. But let us recall that the oracle strengthens the computing power only if it is (at least) capable of solving the halting problem. Similar observations can be made about Petri nets or interactive proof systems, which are other theoretical descriptions of distributed and interactively communicating processes. So we conclude that if the interaction of a system with an external environment gives hypercomputable effects, their source is located outside the analysed system – not in their (interactive) structure. None of these models – important from the point of view of computational complexity – lead to the strengthening of computational power by interaction only (new possibilities are effects of uncomputable information obtained from external sources).

The infinite process of cyclical repetition of certain calculations with periodical output of partial results is known as long ago as the art of programming. It is easily simulated on a standard Turing machine.

In both cases, the problem is that if all elements of our construction remain – whether using cyclic or interacting computations – within limits of classical computability, then we will get only such results that are within possibilities of classical computational models. In order to break through the hypercomputation barrier there is a need of some system component which itself goes beyond the limits of normal computability. So, it turns out that it is not the interaction or infinite loop that introduces a new quality, but an uncomputable function hidden in the background. In this way, instead of new solutions, we simply get the postulate of the possibility of using uncomputable processes in algorithms.

From the practical point of view, Burgin's descriptions do not provide a sufficient explanation of how information dependent on the whole course of an infinite process could be extracted in a finite time.²⁵

Let us give for illustration an intuitive translation of his solution to the halting problem:

- display a message on the monitor at the beginning of the action ‘the program being tested for the given data has not stopped’,
- then run the program under test for the given data and if it is finished, display the message ‘the program under test for the given data has stopped’.

This simple example is intended to present the possibility of obtaining a response for an infinite process in a finite time. However, it is clear that a negative message is not a real result, because it can be modified. This type of machines could only be useful if somehow writing out a negative message guarantees its finality – which, however, is only possible after an infinite number of steps. In its current form, the Burgin proposal simply mislead the role of a temporary assumption with the final outcome of the process.

4. Does hypercomputation cancel CT?

Returning to a more general perspective, one can see that Burgin presents a variant of Hilbert's research optimism, the motto of which could be the famous saying: ‘Wir müssen wissen. Wir werden wissen’. However, the answer to the key question: ‘can computational practice accomplish postulates of some theory of (hyper)computability?’ must have a stronger justification than believing in unlimited progress.

A more general question may relate to the connection between the (not at all obvious) progress of human thought and computations. The power (and its limitations) of computability can influence a vision of admissible

scientific (and especially mathematical) research practices. However, classical computability maybe out of place in this field. Already in the Turing's lecture from 1947, (cf. Turing (1986), p. 393–394) we can find the following observations:

... there can be no machine, which will distinguish provable formulae of the system from unprovable (...) On the other hand if a mathematician is confronted with such a problem he would search around and find new methods of proof, so that he ought eventually to be able to reach a decision about any given formula.

This fragment suggest that a real scientific progress is based on human creativity, which is often seen in mathematics as a special kind of (mathematical) intuition. Is such intuition based on the use of some infinite resources or activities of the human mind?²⁶ Can this infinity be reproduced in hypercomputable processes?²⁷

The analyses made by Burgin do not eliminate this problem of the uncomputable use of infinite resources. This brings us back to the question of whether the practice can make any real use of the infinite processes at all? CT says that this is not possible because the concept of “efficiency” includes the requirement to use only such resources (time, memory, etc.) which can be efficiently employed – as opposed to allowing actual infinite resources to be used in calculations. Burgin, without paying attention to this, suggests that engineering actions should be taken despite the lack of any specific descriptions how well-known physical principles should be used to obtain such extraordinary results. One may wonder to what extent his program is not a proposal for the construction of some *perpetuum mobile* for computer science.

It is worth to observe that Burgin's description (cf. Burgin (2005), p. 27) of the concept of an algorithm says that ‘[r]egules forming the algorithm are unambiguous, simple to execute (implement) and have simple, finite descriptions.’ This formulation of the concept of effectiveness, as a feature of being simple to implement, is rather unusual. If we relate it to functions, then we would have to talk about functions that are effectively computable, as ones whose calculations are easy to perform, which seems to be inconsistent with the intended meaning of the term.

The recurring theme of hypercomputation considerations is the use (open or hidden) of infinite resources of some kind to break through the barrier of uncomputability. In this context, it is worth to recall the important result given by Shoenfield Lemma. This lemma says that a relation is of Δ_2 degree if and only if it can be considered to be the limit of some

sequence of values of a recursive (decidable) relation. The result reinforces the conviction that in order to obtain a function with computational power that exceeds partially recursive functions (so, of the degree Δ_2 or higher), one must use the entire infinite sequence of values. Their implementation would have to guarantee that the actual infinity can be captured through physical mechanisms operating with finite resources.

CT basically says that obtaining an uncomputable calculation is only possible if we give up the efficiency requirement (i.e., also the finite resources requirement). Thus, it can be concluded that CT implies the conviction that in the world of physics it is not possible to obtain an effective result depending on an essentially infinite process using only finite resources (whose elements do not have essentially infinite characteristics). Most of the proposals made by the supporters of hypercomputations do not, as yet, explain how the infinity barrier could be broken in practice. This shows that so far nothing – apart from the rather vague remarks – occurred that would falsify CT, and its role in determining the limits of realistic computations possibilities is still fundamental.

As can be seen from the survey of the different ideas of hypercomputable processes, these studies have some common features, which we try to describe below.

A. They mainly concern Turing's thesis (TT) and not Church's thesis (CT). As indicated earlier, we distinguish between the two theses. The important difference is that TT refers (almost directly) to machine, mechanical calculations, while CT refers to the computability by the mind of a man.

B. Research on hypercomputation goes mainly in two directions:

1. on the one hand, we have some modifications of the concept of Turing machine, i.e. on the formal side of TT. These manipulations have different forms:
 - (a) they may concern inner structure of Turing machines,
 - (b) they may concern additional resources attached to TM,
2. on the other hand, we have a different definition of the intuitive part of TT, which in fact results in the formulation of a new, different thesis.

C. There is some conviction (suggested in Kreisel's remarks on CT) that in the case of classical computability there is a systemic error. It consists in the fact that all (traditional) models are mutually related – so their similarity weakens a conviction of correctness of a classical notion of computation. But in the same manner we can see some systematic error in ideas of hypercomputation. Many works criticise the concept of computability contained in the formulations of CT and TT and try to ex-

pand it.²⁸ It is not a problem in itself to transfer the term computability to certain processes that somehow exceed TM (just as there is no problem with the use of the same term “number” in relation to different classes of mathematical objects: complex numbers, ordinal numbers or quaternion numbers). It should be noted, however, that such supporters of hypercomputations forget that CT refers to effective computability and not simply to computability itself.

So, let’s analyse the problem of efficiency a little more closely. The phrase ‘effectively computable function’ can be presented as $EC(f)$, using the following notation: E – effective; C – computable and f – function in natural numbers. It would seem to happen that $EC(f) \leftrightarrow (E \wedge C)(f) \leftrightarrow E(f) \wedge C(f)$. If it were the case, then $EC(f) \rightarrow E(f)$, but this is not obvious. The argument for rejecting the above implication comes from a linguistic analysis. In reference to the Merriam-Webster dictionary²⁹ effective means something which creates well-defined, final and intended effect. With this understanding of the term, the considered implication $EC(f) \rightarrow E(f)$ becomes false when we take (as some example) the Ackermann function A as the argument f . The function A is a recursive function, so formally falls within the class of functions considered to be effectively computable. However, the Ackermann function is practically ineffective due to the rapid growth, which makes it impossible to calculate effectively for already relatively small (two-digit) arguments.

It follows from the above example that $EC(f)$ should be understood as: $(C[E])(f)$, where E is a modifier of the C predicate. From this we can conclude that in most cases hypercomputation is some form of computability (it is $C[X]$ for some X), but it does not meet the efficiency condition: $X \neq E$, i.e. X cannot be considered as effective in terms of practical use.

D. Finally, it is worth noting that Mike Stannett (cf. Stannett (2001)) gave an argument for the fact that hypercomputation is not experimentally discernible and therefore not falsifiable. Using the criterion of being the proper scientific theory in Popper’s sense, hypercomputability should be considered unscientific. This is in an agreement with the view of M. Davis (Davis, 2006). There is no room for the analysis of Stannett’s position here, but it cannot be ignored.

To sum up briefly, it can be seen that research on the hypercomputations does not invalidate problems related to CT. It seems, therefore – contrary to Burgin’s hasty and pessimistic diagnosis of the ‘fall’ of Church’s thesis – that CT is, for all practical purposes, quite well and still serving its purpose: it delineates the limits of what a mind of *homo sapiens* can effectively calculate.

N O T E S

* The paper is based on the following text: A. Olszewski and J. Mycka, "Czy teza Churcha ma jeszcze jakieś znaczenie dla informatyki", [in:] *Informatyka a filozofia. Od informatyki i jej zastosowań do światopoglądu informatycznego*, 53–74, Wydawnictwo Politechniki Warszawskiej, (2015).

¹ The above remark is motivated by the need to refer to rapidly changing computer technologies.

² It's a little bit more complex. The first statement of Burgin's work is: 'Church-Turing's thesis is false', the second is that the concept of calculability is not absolute, but relative and dependent on available resources (cf. (Copeland and Sylvan (1999), pp. 1–2). It is worth noting that even if the notion of calculability is relative, the relativizations of CT to particular types of calculations and resources are important questions, and in no way can falsity of CT be prejudged.

³ Both terms: 'calculable' and 'computable' will be used interchangeably in this text.

⁴ We pay special attention here to the work of Mark Burgin. We treat it as a clear representation of a certain type of a challenge for CT.

⁵ The first chapter of an interesting anthology (Murawski, 2014) of important texts on computer science given in Polish translation, edited by R. Murawski, is devoted to texts containing explanations of what computer science is in its essence; of course (Harel, 1992) is also worth reading.

⁶ This is a very general characteristic. We are not going into details, because we are interested in different aspects.

⁷ This is what Burgin, for example, believes (see Burgin (2005), p. 14).

⁸ It can be formulated: $EF(id)REC$ or $EF(id)TUR$, where: EF is a concept of an effectively calculable function, REC is a concept of a recursive function, TUR is a concept of a calculable function in the sense of Turing machines.

⁹ This relationship has the usual properties of identity and we mark it with the abbreviation *id*.

¹⁰ The MEP is in a way a consequence of the CT in the formulation of Church himself.

¹¹ Although they are based on absolute geometry, as its special cases. The question then arises whether absolute geometry is realized or realisable in reality?

¹² Here the laws of physics are treated as operational models allowing for an effective interaction with the physical world (not as absolute truths).

¹³ Copeland and Shagrir try to attack the 'proof' of the mechanistic version of the CT given by Gandy (cf. Copeland and Shagrir (2007)).

¹⁴ Such TMs are regarded by some as the first hypercomputers.

¹⁵ We will come back to this issue later in the work.

¹⁶ However, such changes may be significant with regard to complexity (cf. Odifreddi (1989), pp. 50–51).

¹⁷ The models of such processes are e.g. Petri nets, the Actor model, parallel RAM.

¹⁸ This is not considered controversial. It is certainly an extension of the usual concept of calculability (cf. Ord (2006), p. 150).

¹⁹ In fact, we are dealing here with computations where input and output are rational numbers – the only difference from the basic model is the possibility to increase the precision of the calculations.

²⁰ It has been proven that an algorithm solving halting problem exists if Turing machines can use some properties of theory of relativity (Scott Aaronson, Mohammad Bavarian, Giulio Gueltrini, *Computability Theory of Closed Timelike Curves*, ‘arXiv [quant-ph]’, September 18, 2016, arXiv:1609.05507) or quantum mechanics with advice (Scott Aaronson, PDQP/qpoly = ALL, ‘arXiv [quant-ph]’, May 22, 2018, arXiv:1805.08577).

²¹ This concept of neural networks, especially in H. Siegelman’s version, was criticized by M. Davis in Davis (2004).

²² In Ord (2006) there is a list of the following types of hypercomputation: a) o-Machines, b) Turing Machines with Initial Inscriptions, c) Coupled Turing Machines, d) Asynchronous Networks of Turing Machines, e) Error Prone Turing Machine, f) Probabilistic Turing Machines, g) Infinite State Turing Machines, h) Accelerated Turing Machines, i) Infinite Time Turing Machines, j) Fair Non-Deterministic Turing Machines.

²³ Next to the names of the creators of the relevant concept, the year of its creation is indicated. A more detailed and critical discussion of Burgin’s concepts is presented in the paper (Burgin (2005), p. 122).

²⁴ Most of the examples discussed earlier came from this group.

²⁵ We are not discussing here possibilities of using the laws of physics to build some hypercomputable devices (still debatable) – we refer to the conviction that current technologies, when properly used, already give hypercomputable results.

²⁶ In this case problems of natural computing should be taken into consideration e.g. (MacLennan, 2004).

²⁷ Interesting discussions related to these problems can be found in (Copeland, 2008) and (Chaitin, 2006).

²⁸ Some people think that we are dealing here with some development, but there are also those who think that hypercomputation is in fact a primitive notion, while Turing (classical) computability is its idealised, borderline case (cf. Stannett (2006), p.5).

²⁹ The Merriam-Webster dictionary (on-line) gives as an explanation for – effect – (we leave only parts important for our deliberations):): 1. a change that results when something is done or happens; 2. an event, condition, or state of affairs that is produced by a cause; 3. a particular feeling or mood created by something; 4. an image or a sound that is created in television, radio, or movies to imitate something real. The same dictionary gives for – effective – (again we leave only the terms important for our deliberations) the following descriptions: 1. producing a result that is wanted; 2. having an intended effect of a law, rule, etc.; 3. in use; 4. starting at a particular time. More detailed definition of effective: 1. producing a decided, decisive, or desired effect (an effective policy); 3. ready for service or action (effective manpower); 4. actual (the need to increase effective demand for goods); 5. being in effect: operative (the tax becomes effective next year).

REFERENCES

- Burgin, M. (2002). The Rise and Fall of Church-Turing Thesis. draft.
- Burgin, M. (2005). *Super-Recursive Algorithms*. Springer.
- Chaitin, G. J. (2006). The Omega Number: Irreducible Complexity in Pure Math. In Borwein, J. M. and Farmer, W. M., editors, *Mathematical Knowledge Management, 5th International Conference, Wokingham, UK*, volume 4108 of *Lecture Notes in Computer Science*. Springer.

- Copeland, J. (2008). The Mathematical Objection: Turing, Gödel and Penrose on the Mind. draft.
- Copeland, J. and Shagrir, O. (2007). Physical Computation: How General are Gandy's Principles for Mechanism? *Minds and Machines*, 17:217–231.
- Copeland, J. and Sylvan, R. (1999). Beyond the universal Turing machine. *Australasian Journal of Philosophy*, 77(1):46–66.
- Davis, M. (2004). The myth of Hypercomputation. draft.
- Davis, M. (2006). Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178:4–7.
- Harel, D. (1992). *Algorithmics*. Pearson Education. In Polish: Harel, David, Rzecz o istocie informatyki. Algorytmika, WNT 2000.
- Hewitt, C. (2013). What is Computation? Actor Model versus Turing's Model. draft.
- MacLennan, B. J. (2004). Natural computation and non-Turing models of computation. *Theoretical Computer Science*, 317:115–145.
- Murawski, R., editor (2014). *Filozofia informatyki*. Wydawnictwo Naukowe UAM.
- Odifreddi, P. (1989). *Classical Recursion Theory*. North-Holland Publishing Company.
- Ord, T. (2002). Hypercomputation: computing more than the Turing machine. draft.
- Ord, T. (2006). The many forms of hypercomputation. *Applied Mathematics and Computation*, 178:143–153.
- Shagrir, O. (2004). Super-tasks, accelerating Turing machines and uncomputability, *Theoretical Computer Science*, 317:105–114.
- Shagrir, O. and Pitowsky, I. (2003). Physical Hypercomputation and the Church-Turing Thesis. *Minds and Machines*, 13:87–101.
- Spaan, E., Torenvliet, L., van Emde Boas, P. (1989). Nondeterminism, Fairness and a Fundamental Analogy, *EACTS Bulletin*, 37:186–193.
- Stannett, M. (2001). Hypercomputation is Experimentally Irrefutable, draft.
- Stannett, M. (2006). The case for hypercomputation, *Applied Mathematics and Computation*, 178:8–24.
- Turing, A. (1986). In Carpenter, B. E. and Doran, R. W., editors, *A. M. Turing's ACE report of 1946 and other papers*, chapter Lecture to the London Mathematical Society on 20 February 1947. MIT Press.
- Zenil, H. (2012). In Zenil, H., editor, *A Computable Universe. Understanding and Exploring Nature as Computation*, chapter ntroducing the Computable Universe. World Scientific Publishing Company.