Technical Report 2013-608
WHAT IS COMPUTATION?

Selim G. Akl
School of Computing
Queen's University
Kingston, Ontario K7L 3N6
Canada
Email: `akl@cs.queensu.ca`

October 3, 2013

Abstract

Three conditions are usually given that must be satisfied by a process in order for it to be called a computation, namely, there must exist a finite length algorithm for the process, the algorithm must terminate in finite time for valid inputs and return a valid output, and finally the algorithm must never return an output for invalid inputs. These three conditions are advanced as being necessary and sufficient for the process to be computable by a universal model of computation. In fact, these conditions are neither necessary, nor sufficient.

On the one hand, recently defined paradigms show how certain processes that do not satisfy one or more of the aforementioned properties can indeed be carried out in principle on new, more powerful, types of computers, and hence can be considered as computations. Thus the conditions are not necessary.

On the other hand, contemporary work in unconventional computation has demonstrated the existence of processes that satisfy the three stated conditions, yet contradict the Church-Turing Thesis, and more generally, the principle of universality in computer science. Thus the conditions are not sufficient.

1 Introduction

SUM ERGO COMPUTO
(I am therefore I compute)
Motto of the School of Computing,
Queen's University

What does it mean "to compute"? This is a question paramount in computer science, for *computation* is the essence of our field. Having a solid definition of the process of computation would ensure that the discipline rests on a firm foundation. However, such definition is lacking and there is no sign of one being agreed upon. Many see the absence of a definition as a weakness, others as a strength. To illustrate, consider the following questions:

1. Is computation the process of performing arithmetical calculations?
2. Do logical operations qualify as computations?
3. What about text, image, and sound processing?
4. Is reading an input from the outside world (for example, by transforming a physical quantity such a temperature into a number expressed, again for example, as a binary string) a computation?
5. Is producing an output that operates on the outside world (like a thermostat controlling temperature in a room) a computation?
6. Or is computation limited to what a particular computational model [38, 41, 52] (such as the Turing Machine, the Random Access Machine, the Cellular Automaton, and so on) is capable of doing?
7. Perhaps computation is reduced to the evaluation of recursive functions [29]?
8. Can the neurons in our brain be said to compute [12]?
9. More unconventionally, are physical phenomena computations? Can a chemical reaction be considered a computation? What about cell multiplication, DNA replication, or electron spin, do they qualify?
10. Is computation present in myriad natural occurrences [17], from reproduction in ciliates [28] to quorum sensing in bacterial colonies [34], from respiration [9] and photosynthesis [47] in plants to the migration of birds [13] and fish [51], and from morphogenesis [10] to foraging for food [1] to human cognition [46, 35]?
11. Are *all* of the above computations? If not, which ones are not, and why?
12. More importantly, if indeed all these can be viewed as computations, is the list exhaustive?

The lack of a strict definition of what it means to compute, is in fact a blessing in disguise, for it is a witness to the vitality and depth of computer science, and its ever evolving nature.

2 The Church-Turing Thesis

"It can also be shown that any computation
that can be performed on a modern-day digital computer
can be described by means of a Turing machine."
John E. Hopcroft and Jeffrey D. Ullman,
Formal Languages and their Relation to Automata
Addison-Wesley, 1969

The founders of the field of computer science, all of them mathematicians (namely, Turing, Church, Kleene, Gödel, Post, von Neumann, and others), felt that computation is equivalent to function evaluation. Specifically, for a function f: Given x, calculate $f(x)$. It was a simple and tight definition. And since the (then) recently-invented Turing Machine did just that (specifically, evaluate functions over the integers), the definition was not only convenient, but also persuasive.

However, the founders, and their successors, knew better. When they decided to make a statement about what it means to compute, they did not formulate it as a *theorem*, but rather as a *conjecture*. Hence was born the famous Church-Turing *Thesis*, according to which everything computable can be computed by a Turing Machine. This interpretation of the Church-Turing Thesis, given in the quote above, continues to be the one most widely adopted by today's computer scientists (see, for example, [22, 26]).

For our purposes in this paper, we note here this important detail: The statement is a *thesis* and not a theorem exactly because of the difficulty of defining precisely what it means to compute.

3 The Three Conditions

Notwithstanding its status as a conjecture, the Church-Turing Thesis is generally believed to be true by the majority of computer scientists. This belief stems from two observations:

1. All commonly known computations to date, no matter how complex, can be executed (in principle) on a Turing Machine.

2. All attempts to define extensions and variants of the Turing Machine, and all attempts to define new models of computation (such as, for example formal grammars, or recursive functions), end up with models that are equivalent to the basic Turing Machine.

A 'belief' is not a proof, however, and no one would seriously propose to define 'to compute' as 'what the Turing Machine does', regardless of how 'universal' the Turing Machine appears to be. Such a definition would turn the Church-Turing Thesis into a tautology, which is clearly not very helpful. (As it turns out, both of the aforementioned observations will be shown to be incorrect later in this paper.)

In an attempt to characterize what a computation is, as generally as possible and independently of models, the following three properties are often used (see, for example, [21, 36]):

1. There exists a finite length algorithm to carry out the computation.
2. For valid inputs, the algorithm terminates in unbounded but finite time, using unbounded but finite space, and returns a valid and correct output.
3. For invalid inputs, the algorithm always fails to return an output; it either hangs, or runs forever, never terminating.

These three conditions are advanced as being necessary and sufficient conditions for a process to be computable by a universal model of computation, such as, for example, the Turing Machine [16, 45].

4 Consequences: Computable And Uncomputable

The three conditions in section 3 allow us to divide processes into two classes, those that are computable and those that are uncomputable. Thus:

1. Those processes that satisfy the three conditions are computations: They are computable by a universal model of computation.
2. Those processes that do not satisfy one or more of the conditions are not computations: They are not computable by any model of computation.

For example, the process of sorting a sequence of items on which a linear order is defined, satisfies the three conditions. Sorting, therefore, is computable.

By contrast, there does not exist an algorithm for determining in finite time whether an arbitrary program will terminate when operating on an arbitrary input. Thus, the Halting Problem, as it is widely known, is uncomputable.

The implication here is that the three conditions are necessary and sufficient for a process to be a computation.

In the remainder of this paper we show that the three conditions in section 3 are neither necessary nor sufficient.

5 Counterexamples To Necessity

Over the last several years, an increasing number of computer scientists began to question the necessity of the three conditions of section 3. The question they posed is: Can a process violate one or more of the conditions and still be called a computation? Their work suggests that the answer is affirmative. We show two examples of their results.

5.1 Hypercomputation: How To Compute The Uncomputable

"Might not a man's skill increase so fast
that he performed each operation
in half the time required for its predecessor?
In that case, the whole infinite series
would take only twice as long as the first operation."
Bertrand Russell,
The Limits of Empiricism,
Proceedings of the Aristotelian Society, 1935

The new field of study known as hypercomputation aims to demonstrate how a change in our thinking about computation leads us to solve problems previously thought to be unsolvable [42]. Consider, for instance, the Halting Problem. It is said to be uncomputable because it violates the first condition in section 3.

However, the problem can be readily solved by an *accelerating machine*, that is, a machine that doubles its speed at every step. Given a program and its input, the accelerating machine executes the program on its input, performing the first instruction in one second, the second in one-half of a second, the third in one quarter of a second, and so on. After exactly two seconds the question as to whether or not the program terminates on its input is settled [14, 23]. Proposals for implementing accelerating machines, as well as several other hypercomputational approaches to solving the Halting Problem, including quantum and relativistic ones, are reviewed in [3].

5.2 Nonterminating Processes

Another school of thought concerns itself with processes that violate the second condition in section 3: Termination on a valid input. Many processes, it is pointed out, never terminate but rightly qualify as computations. Examples include operating systems, monitoring systems, the Internet, and so on [11, 18, 25, 48]. These are clearly computations yet they violate the termination condition.

The conditions of section 3 are therefore not necessary for a process to be called a computation.

6 Nonuniversality: A Counterexample To Sufficiency

It is possible for a process to be eminently computable, but not computable by a 'universal' computer. This is not a paradox. We prove in this section that there exist processes that satisfy the three conditions of section 3, and yet cannot be computed (neither in theory nor in practice) by a computer that is fixed once and for all.

Here, a *time unit* is defined as the time it takes the fastest available processor to perform a constant number of elementary operations (such as basic arithmetic and logic operations, pairwise exchange, and so on).

A general formulation of the nonuniversality result is as follows. Let U be a putative universal computer. By definition of a universal computer, U is capable of exactly $T(i)$ operations during time unit i, where i is a positive integer and T is a function of i, such that for each i, $T(i)$ is finite and fixed once and for all. Note that $T(i)$ may be a constant (as for the Turing Machine), or it may vary with i (as for the accelerating machine), but it must be specified in advance and unchangeable. It follows that U cannot be universal, for there exists at least one computation that U cannot perform, namely, a computation requiring $V(i)$ operations during time unit i, where $V(i) > T(i)$ for at least one i. Therefore, no computer is universal. [3] – [8], [32], [33].

In the remainder of this section we present two computational problems, each of which serving as a counterexample to the existence of a universal computer, and consequently to the sufficiency of the conditions in section 3.

6.1 Computing Under Mathematical Constraints

"Computation is much like life.
The beginning is known
and the end is more or less predictable
and rather uninteresting.
More important, and far more exciting,
is what happens between the start and finish,
and how it happens; that's what matters."
Josh Mozersky,
Department of Philosophy, Queen's University,
Personal communication, 2013

In order to disprove sufficiency, we define two problems that belong to a class of computations in which each step must obey a mathematical constraint [32]. In these computations, the process of arriving at the final result is no less important than the result itself. In other words, here the journey counts as much as the destination. While it is true that we are interested in the outcome of the computation (this being the *destination*), for this class we are more concerned with *how* the result is obtained (namely, there is a constraint that must be satisfied throughout all steps of the algorithm, this being the *journey*). It is worth emphasizing in this context that the constraint to be satisfied is germane to the problem itself; specifically, there are no restrictions whatsoever on the model of computation or the algorithm to be used. Our task is to find an algorithm for a chosen model of computation that solves the problem exactly as posed. One should also observe that computer science is replete with problems with an inherent constraint on how the solution is to be obtained. Examples of such problems include: Inverting a nonsingular matrix without ever dividing by zero, finding a shortest path in a graph without examining an edge more than once, sorting a sequence of numbers without reversing the initial order of equal inputs, and so on.

6.1.1 Generating Combinatorial Objects

Our first existence proof uses the process of generating all instances of a combinatorial object, such as permutations, combinations, derangements, and so on. Specifically, consider the sequence $S = \langle s_1, s_2, \ldots, s_n \rangle$ of n elements, where $n \geq 2$. A total order $\prec$ is defined on the elements of S such that $s_1 \prec s_2 \prec \cdots \prec s_n$. Now suppose that it is required to generate all $n!$ permutations of the sequence S, consecutively in a register R of n positions, where each position of R stores one element of S. For our purposes, the following constraint must be satisfied by any algorithm that generates the permutations of S: At the end of each time unit, R must contain the next permutation in *minimal-change order*, that is, each permutation is to differ from the previous one in the least possible way.

This is a nontrivial problem, not only for a sequential (that is, single processor) computer, but even for a parallel computer equipped with n processors, for it is required that each permutation of the n elements of S be produced in R in constant time (specifically in *one* time unit–a quantity independent of n), as well as follow the previously generated one in minimal-change order. There exists an algorithm that executes the required process on a linear array of n processors [2]. Any attempt to solve the problem using fewer than n processors fails, as it cannot satisfy the constraint of having a new permutation appear in R every time unit, and in minimal-change order to boot. Thus while the process is computable, it is not computable by a machine that is fixed once and for all, as a 'universal' computer would be [3] – [8].

6.1.2 Sorting With A Twist

A second example of computations obeying a mathematical constraint is provided by a variant to the problem of sorting. For a positive even integer n, where $n \geq 8$, let n distinct integers be stored in an array A with n locations $A[1]$, $A[2]$, …, $A[n]$, one integer per location. Thus $A[j]$, for all $1 \leq j \leq n$, represents the integer currently stored in the jth location of A. It is required to sort the n integers in place into increasing order, such that:

1. After step i of the sorting algorithm, for all $i \geq 1$, no three consecutive integers satisfy:

$$A[j] > A[j+1] > A[j+2]$$

   for all $1 \leq j \leq n-2$.

2. When the sort terminates we have:

$$A[1] < A[2] < \cdots < A[n].$$

This is the standard sorting problem in computer science, but with a twist. An algorithm for an $n/2$-processor parallel computer solves the problem handily by means of pairwise swaps applied to the input array A, during each of which $A[j]$ and $A[k]$ exchange positions (using an additional memory location for temporary storage). A sequential computer, and a parallel computer with fewer than $(n/2) - 1$ processors, both fail to solve the problem consistently, that is, they fail to sort all possible $n!$ permutations of the input while satisfying, at every step, the constraint that no three consecutive integers are such that $A[j] > A[j+1] > A[j+2]$ for all j. In the particularly nasty case where the input is of the form

$$A[1] > A[2] > \cdots > A[n],$$

any sequential algorithm and any algorithm for a parallel computer with fewer than $(n/2) - 1$ processors fail after the first swap [33].

It is interesting to note here that a Turing Machine with $n/2$ heads succeeds in solving the problem, yet its simulation by a standard (single head) Turing Machine fails to satisfy the requirements of the problem. Indeed, suppose that the standard Turing Machine is presented with the input sequence $A[1] > A[2] > \cdots > A[n]$:

1. It will either use the given representation of the input, and proceed to perform an operation (a swap, for example), in which case it would fail after one step of the algorithm,

2. Or it will transform the given representation into a different encoding (perhaps one intended to capture the behavior of the Turing Machine with $n/2$ heads) in preparation for the sort, in which case it would again fail since the transformation itself constitutes an algorithmic step.

In itself, this is a surprising result, as it goes against the common belief mentioned in section 3 that any computation by a variant of the Turing Machine can be effectively simulated by the standard model [29].

Thus, the two examples of this section prove that the conditions of section 3 are not sufficient for a process to be computable by a universal computer.

7 So, What *Is* Computation?

"I shall not define macrosociology;
instead I shall briefly describe a number of studies
in order to draw out the basic procedures
they seem to have in common.
The examples will be taken from a list which ...
may serve as a temporary implicit definition of the field."
Paul F. Lazarsfeld,
On Social Research and its Language,
University of Chicago Press, 1993

"I know it when I see it."
Potter Stewart,
Jacobellis v. Ohio,
United States Supreme Court, 1964

In the second decade of the twenty-first century, it is abundantly clear that computation is too rich, too ubiquitous, and too fundamentally important an idea to be confined to the restrictive straitjacket of inept definitions such as "Computation is what the Turing Machine does", or "Computation is function evaluation", and so on. It is also clear that no narrow model-bound definition will do, as new models are continuously being defined in order to capture newly discovered computational phenomena.

One might choose to be content with a definition by cases: A collection of processes, accepted as computations, could be used. This collection will no doubt grow indefinitely. There is nothing wrong with this. Many fields of knowledge use this approach.

However, definitions play a crucial role in an exact science. They provide a standard that unifies and guides the scientific enterprise. And while an all encompassing definition of what it means to compute is not forthcoming, a working definition is needed that is general enough to apply to all of today's computations, as well as anticipated ones. Hence we venture to propose one such definition here.

The most general definition of computation is as *information processing.* It is the process of acquiring information from the outside world, transforming it, and providing the outcome to the outside world. This definition of information processing as a series of three step sequences consisting of input, data manipulation, and output applies equally well to computations initiated and controlled by human beings, as well as those occurring spontaneously in Nature without human intent or intervention.

8 Conclusion

"In a sense Nature has been continually computing
the 'next state' of the universe for billions of years;
all we have to do - and actually, all we can do -
is 'hitch a ride' on this huge ongoing computation."
Tommasso Toffoli,
Physics and Computation,
International Journal of Theoretical Physics, 1982

"Think of all our knowledge-generating processes,
our whole culture and civilization,
and all the thought processes
in the minds of every individual,
and indeed the entire evolving biosphere as well,
as being a gigantic computation.
The whole thing is executing a self-motivated,
self-generating computer program."
David Deutsch,
The Fabric of Reality,
Penguin Books, 1997

Once upon a time, biologists thought that, thanks to their knowledge of cells, they had a description of how the world works. And then chemists came along and told us that everything happens at the molecular level. When physicists arrived on the scene, they said no, all the action is at the atomic level. Today, computer scientists are thinking that perhaps things can be explained at the bit level. That is, at the information level.

It may very well be the case that, ultimately, we will have to adopt the point of view, espoused already by many, that at the very bottom everything is indeed a computation [15, 19, 20, 24, 27, 30, 37, 39, 40, 43, 44, 49, 50, 52, 53]. Computation permeates the Universe and drives it. Every atom, every molecule, every cell, everything, everywhere, at every moment is performing a computation. The Universe is a giant computer. In the end, it may very well be the case that, far from being a human *invention*, computation will turn out to be another magnificient human *discovery*!

References

[1] Adamatzky, A. and Akl, S.G., Trans-Canada slimeways: Slime mould imitates the Canadian transport network, *International Journal of Natural Computing Research*, Vol. 2, Issue 4, October-December 2011, pp. 31–46.

[2] Akl, S.G., *Parallel Computation: Models and Methods*, Prentice Hall, Upper Saddle River, New Jersey, 1997.

[3] Akl, S.G., Three counterexamples to dispel the myth of the universal computer, *Parallel Processing Letters*, Vol 16, No. 3, 2006, pp. 381–403.

[4] Akl, S.G., Even accelerating machines are not universal, *International Journal of Unconventional Computing*, Vol. 3, No. 2, 2007, pp. 105–121.

[5] Akl, S.G., Evolving computational systems, Chapter One in: S. Rajasekaran and J.H. Reif, Eds., *Handbook of Parallel Computing: Models, Algorithms, and Applications*, Taylor and Francis, CRC Press, Boca Raton, Florida, 2008, pp. 1–22.

[6] Akl, S.G., Unconventional computational problems with consequences to universality, *International Journal of Unconventional Computing*, Vol. 4, No. 1, 2008, pp. 89–98.

[7] Akl, S.G., Time travel: A new hypercomputational paradigm, *International Journal of Unconventional Computing*, Vol. 6, No. 5, 2010, pp. 329 - 351.

[8] Akl, S.G., Nonuniversality in computation: Thirteen misconceptions rectified, Technical Report No. 2013-609, School of Computing, Queen's University, August 2013. `http://www.cs.queensu.ca/home/akl/techreports/thirteen.pdf`

[9] Ball, P., Do plants act like computers?, *Nature*, January 2004, `http://www.nature.com/news/1998/040119/full/news040119-5.html`

[10] Bhattacharyya, A., Morphogenesis as an amorphous computation, *Proceedings of the Third ACM Conference on Computing Frontiers*, Ischia, Italy, May 2006, pp. 53–64.

[11] Burgin, M., *Super-Recursive Algorithms*, Springer, New York, 2005.

[12] Cazé, R.D., Humphries, M., and Gutkin, B., Passive dendrites enable single neurons to compute linearly non-separable functions, *PLoS Computational Biology*, Vol. 9, No. 2, February 2013.

[13] Connor, S., Revealed: Secret of how birds navigate during migration, *The Independent*, May 1, 2008,
`http://www.independent.co.uk/news/science/revealed-secret-of-how-birds-navigate-d`
`html`

[14] Copeland, J., Even Turing machines can compute uncomputable functions, in: C. Calude, J. Casti, and M. Dinneen, Eds., *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998, pp. 150–164.

[15] Davies, E.B., Building infinite machines, *British Journal for Philosophy of Science*, Vol. 52, 2001, pp. 671–682.

[16] Davis, M., *The Universal Computer*, W.W. Norton, New York, 2000.

[17] De Castro, L.N., *Fundamentals of Natural Computing: Basic Concepts, Algorithms , and Applications*, Chapman & Hall/CRC, New York, 2006.

[18] Denning, P.J., Reflections on a Symposium on Computation, *Ubiquity*, ubiquity.acm.org; see also *The Computer Journal*, Vol. 55, No. 7, 2012, 799–802.

[19] Deutsch D., *The Fabric of Reality*, Penguin Books, London, England, 1997.

[20] Durand-Lose, J., Abstract geometrical computation for black hole computation, Research Report No. 2004-15, Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, Lyon, France, April 2004.

[21] Enderton, H.B., Elements of recursion theory, in: J. Barwise, Ed., *Handbook of Mathematical Logic*, North-Holland, Amsterdam, 1977, pp. 527–566.

[22] Fortnow, L., The enduring legacy of the Turing machine, *The Computer Journal*, Vol. 55, No. 7, 2012, pp. 830–831.

[23] Fraser R. and Akl, S.G., Accelerating machines: A review, *International Journal of Parallel Emergent and Distributed Systems*, Vol. 23, No. 1, February 2008, pp. 81–104.

[24] Gleick, J., *The Information: A History, a Theory, a Flood*, HarperCollins, London, England, 2011.

[25] Goldin, D. and Wegner, P., The Church-Turing thesis: Breaking the myth, *Proceedings of the First international conference on Computability in Europe: new Computational Paradigms*, Springer-Verlag, Berlin, 2005, pp. 152–168.

[26] Hein, J.L., *Discrete Structures, Logic, and Computability*, Jones and Bartlett, Subury, Massachusetts, 2010, p. 89.

[27] Kelly, K. God is the machine, *Wired*, Vol. 10, No. 12, 2002.

[28] Kari, L. and Rozenberg, G., The many facets of natural computing, *Communications of the ACM*, Vol. 51, No. 10, 2008, pp. 72–83.

[29] Lewis, H.R. and Papadimitriou, C.H., *Elements of the Theory of Computation*, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[30] Lloyd, S., *Programming the Universe*, Knopf, New York, 2006.

[31] Nagy, M. and Akl, S.G., Quantum computing: Beyond the limits of conventional computation, *International Journal of Parallel, Emergent and Distributed Systems*, Special Issue on Emergent Computation, Vol. 22, No. 2, April 2007, pp. 123–135.

[32] Nagy, M. and Akl, S.G., Parallelism in quantum information processing defeats the Universal Computer, *Parallel Processing Letters*, Special Issue on Unconventional Computational Problems, Vol. 17, No. 3, September 2007, pp. 233–262.

[33] Nagy, N. and Akl, S.G., Computing with uncertainty and its implications to universality, *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 27, Issue 2, April 2012, pp. 169–192.

[34] Onuchic, J., Bacteria use chat to play the 'prisoner's dilemma' game in deciding their fate, *Science Daily*, March 2012, `http://www.sciencedaily.com/releases/2012/03/120327215704.htm`

[35] Pylyshyn, Z.W., *Computation and Cognition*, MIT Press, Cambridge, Massachusetts, 1984.

[36] Rogers, H. Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.

[37] Rucker, R., *The Lifebox, the Seashell, and the Soul*, Thunder's Mouth Press, New York, 2005.

[38] Savage, J.E., *Models of Computation*, Addison-Wesley, Reading, Massachusetts, 1998.

[39] Seife, C., *Decoding the Universe*, Viking Penguin, New York, 2006.

[40] Siegfried, T., *The Bit and the Pendulum*, John Wiley & Sons, Inc., New York, 2000.

[41] Sipser, M., *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, Massachusetts, 1997.

[42] Syropoulos, A., *Hypercomputation*, Springer, New York, 2008.

[43] Tipler, F.J., *The Physics of Immortality: Modern Cosmology, God and the Resurrection of the Dead*, Macmillan, London, England, 1995.

[44] Toffoli, T., Physics and Computation, *International Journal of Theoretical Physics*, Vol. 21, 1982, pp. 165–175.

[45] A.M. Turing, On computable numbers with an application to the entscheidungsproblem, *Proceedings of the London mathematical Society*, Ser. 2, Vol. 42, 1936, pp. 230–265; Vol. 43, 1937, pp. 544–546.

[46] Van Gelder, T., What might cognition be, if not computation?, *The Journal of Philosophy*, Vol. 92, No. 7., July 1995, pp. 345-381.

[47] Van Hulst, N., Uncovering Quantum Secret in Photosynthesis, *Science Daily*, June 2013, `http://www.sciencedaily.com/releases/2013/06/130620142932.htm`

[48] Van Leeuwen, J. and Wiedermann J., The Turing machine paradigm in contemporary computing, in: B. Engquist and W. Schmidt, Eds., *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2000, pp. 1139–1156.

[49] Vedral, V., *Decoding Reality*, Oxford University Press, Oxford, England, 2010.

[50] Wheeler, J.A., *At Home in the Universe*, American Institute of Physics Press, Woodbury, New York, 1994.

[51] Wolchover, N., A secret to animal migration believed unlocked at last, *Science on NBC News*, July 9, 2012,
`http://www.nbcnews.com/id/48124812/ns/technology_and_science-science/t/secret-animal-migration-believed-unlocked-last/#.UgV5U20piM0`

[52] Wolfram, S., *A New Kind of Science*, Wolfram Media, Champaign, Illinois, 2002.

[53] Zuse, K., *Calculating Space*, MIT Technical Translation AZT-70-164-GEMIT, Massachusetts Institute of Technology (Project MAC), Cambridge, Massachusetts, 1970.