Computational Completeness of Interaction Machines and Turing Machines

Peter Wegner[1], Eugene Eberbach[2] and Mark Burgin[3]

[1] Dept. of Computer Science, Brown University, Providence, RI 02912, USA
peter_wegner@brown.edu
[2] Dept. of Eng. and Science, Rensselaer Polytechnic Institute, Hartford, CT 06120, USA
eberbe@rpi.edu
[3] Dept. of Mathematics, University of California, Los Angeles, CA 90095, USA
mburgin@math.ucla.edu

Abstract

In the paper we prove in a new and simple way that Interaction machines are more powerful than Turing machines. To do that we extend the definition of Interaction machines to multiple interactive components, where each component may perform simple computation. The emerging expressiveness is due to the power of interaction and allows to accept languages not accepted by Turing machines. The main result that Interaction machines can accept arbitrary languages over a given alphabet sheds a new light to the power of interaction. Despite of that we do not claim that Interaction machines are complete. We claim that a complete theory of computer science cannot exist and especially, Turing machines or Interaction machines cannot be a complete model of computation. However complete models of computation may and should be approximated indefinitely and our contribution presents one of such attempts.

Keywords: Turing machines, Interaction machines, computation, completeness, expressiveness

1 Introduction

Alan Turing in his 1936 paper "On Computable Numbers with an Application to the Entscheidungsproblem" showed limits of Turing machines [21]. His further work on choice machines, oracle machines and unorganized machines gave additional evidence, demonstrating that Turing machines (TM) do not form the definite limit of all possible computations [22, 23]. It is therefore surprising that some theoretical computer scientists suggest that Turing machines (TM) form a complete model of computation.

The theory of everything is related to the attempts to construct a complete theory of physics, mathematics, or philosophy, which have been repeatedly attempted, but not realized. Hilbert's Entscheidungsproblem was such an attempt of theory of everything for mathematics, disproved by Gödel [17], Church [10] and Turing [21]. Aristotle, Plato, Hegel wanted to construct all-encompassing systems for philosophy. Newton, Laplace and Einstein attempted a theory of everything for theoretical physics. Einstein searched unsuccessfully for a unifying theory of gravity and electromagnetism. String theories and supergravity are next attempts to construct a complete theory of physics. Hawking several times changed his mind whether a theory of everything for physics is possible at all [18]. He justified his change of mind that "some people will be very disappointed if there is not an ultimate theory that can be formulated as a finite number of principles".

In a similar way, Turing machines have been treated as the ultimate foundation of a "theory of everything" for computer science. This approach was expressed by the strong Church-Turing

thesis, which stated that everything computable is computable by a Turing machine. We understand and sympathize with the desire of many computer scientists to believe that Turing machines is an everlasting complete and final model of computation. We understand also their disappointment when experts prove that their belief is wrong. They do not want to accept this and try to suppress new knowledge.

Real computers continuously evolve, however their predefined model - TM (invented before any real computer had been built)- for all years remained the same. It is probably too much to expect that even so elegant a model as TM, will describe all computers built and ever to be built. It is also unrealistic to expect that TMs will cover all forms of natural computation. TM has been designed to model a single automatic machine, shutting down the world during computation. Now, most computers are interconnected and interact with other computers and its (possibly uncountable) environment. Many programs (e.g., OS, servers, and other reactive systems) do not intend at all to terminate and we do not know their initial state. Additionally, consecutive steps of computation (e.g., as in neural networks) may not have well defined semantics, still they are doing useful things, but violating classical definition of an algorithmic problem solving.

It is correct that many early models of computation turned out to be equivalent to Turing machines. Many extensions of the basic TM model failed to show higher computing power, i.e., they were not more expressive than the basic TM. Computational models from the long list developed for 45 years after the first Turing's paper were either equally expressive or even less expressive than Turing machines. Active research in the theory of algorithms and computation brought forth such models as Herbrand functions, Gödel (recursive) functions, Church λ-calculus, Kleene (partial recursive) functions, Post systems, Markov algorithms, Chomsky unrestricted and context-free grammars, Knuth attribute grammars, and RAM machines. There were also proposed direct extensions of the basic TM model in the form of multitape TMs, nondeterministic TM's, multistack machines, or counter machines, which also have been proven to be functionally equivalent to a single tape deterministic TM. The above models of computation were included in the classical theoretical computer science curriculum and used as the convenient argument to stop any attempts of going beyond the canon of TM.

On the other hand, underneath, many models have been proposed to support the belief that the TM is only one level in the hierarchy of computational models, but not its absolute limit. What is also interesting, several models that are higher than TM in the computational hierarchy are as old as TM. However, a part of the computer science community decided that the TM model gives the absolute limit for the power of computation, and ignored more expressive than classical TMs models. For such people, it was much easier to live in the closed computational universe than to invent or even to learn something new and more relevant. For many years, Turing machines played the role of the absolute limit in computer science similar to the absolute zero in physics or constant speed of light in relativity theory or the unique infinity in mathematics, which is larger than any number. However, this situation changed in mathematics when Cantor proved that there were more real numbers than integers. Turing used the same diagonalization argument to distinguish undecidable problems from decidable ones. After this, Turing machines have been regularly used to prove insolvability of many problems and incompleteness of different systems. It has been always assumed that those results demonstrated limits of people's abilities and deficiency of incomplete systems. However, new discoveries demonstrated that those results displayed only limits of the TM model as a problem solver and decision maker.

Turing himself did not accept his TM model as a complete model of computation. In particular, Turing besides his classical a-machines (known commonly as TMs [21]), proposed

more expressive TM with oracles (o-machines) [22], choice machines (c-machines) [21], and unorganized machines (u-machines) [23]. Ulam and von Neumann in the search of a universal constructor (subsuming universal computability) turned out to the help of expressive power of cellular automata [26] rather than TMs. Several scientists were attracted by expressiveness of neural networks, analog computing, reactive systems, natural computing. The list of models going beyond Turing Machines is quite long, and includes mentioned before o-machines, c-machines, u-machines, Inductive Turing Machines, Limit Turing Machines, Interaction Machines, Persistent Turing Machines, Site and Internet Machines, Infinite Time Turing Machines, Evolutionary Turing Machines, random automata networks, neural networks, π-calculus and \$-calculus (see, e.g., [5, 12, 14, 20, 27, 28, 29]). The chapter [13] in the Festschrift book stressed achievements of Alan Turing in computer science explaining that Alan Turing was the first scientist proposing to go beyond the Turing Machine limit. The whole book [16] was devoted to models based on interaction, explaining why interaction is pivotal for the future of computer science.

In this paper we demonstrate the limitations of Turing machines and classical recursive (terminating) algorithms. We also show how superTuring models of computation may overcome limits of TMs. For that we provide a completely new extended definition of Interaction machines (in fact, we define multiple types of Interaction machines), and for this new extended superTuring model we prove that Interaction machines consisting of conventional components can accept languages that Turing machines cannot accept. Our goal is not to provide a complete overview of models of interaction (we define several such extensions but do not explore them in the paper in depth further; for more details, see, e.g., [16]), but to provide a simple and new way to prove that interaction is more expressive than TMs. Simply, many researchers still question the power and feasibility of new models of computation and claim that Turing machines capture all computations already. In the attempt to convince sceptics, we used Interaction machines that are much simpler than other potential models of interaction. Note that we extended original definitions of Interaction machines, thus this is a new and original contribution. We needed simplicity, because it might provide some hints for feasibility of implementation of superTuring models of computation (e.g., like in reactive systems or by approximation of infinity). However, Interaction machines, although more powerful than Turing machine, are also incomplete. The authors believe that a complete theory of computer science and complete theory of everything will never be achieved, but can and should be approximated.

2 Why Turing Machines Do Not Cover All Computations

In this section, we do not present something unusual. We remind simply that there are classes of languages that cannot be modeled by Turing machines. If we associate computations with corresponding languages (sets of strings over a given finite alphabet), then there are computations that are not possible to express by TM.

Turing machines [21] and algorithms are two fundamental concepts of computer science and problem solving. Turing machines describe the limits of the algorithmic problem solving, and laid the foundation of current computer science in the 1960s. It turns out that *undecidable* problems cannot be solved by TMs and *intractable* problems are solvable, but require too many resources (e.g., steps or memory). For undecidable problems effective recipes do not exist problems are called nonalgorithmic or nonrecursive. On the other hand, for intractable problems algorithms exist, but running them on a deterministic Turing machine, requires an exponential amount of time (the number of elementary moves of the TM) as a function of the TM input.

Simplicity of the TM model allowed mathematicians and computer scientists to formally prove that there are specific problems (languages) that the TM cannot solve (decide or compute). In some cases, solving a problem is equivalent to deciding whether a string belongs to some formal language. That is why a problem that cannot be solved by Turing machine is called undecidable (or more exactly, recursively undecidable). The class of languages accepted by Turing machines is called recursively enumerable (RE-) languages. For RE-languages, TM can accept the strings in the language but cannot tell for certain that a string is not in the language.

There are two classes of Turing machine unsolvable languages (problems):

- *recursively enumerable but not recursive* (*RE non-recursive* *REnon-rec* for short) - TM can accept the strings in the language but cannot tell for certain that a string is not in the language (e.g., the language of the universal Turing machine, or Post Correspondence Problem languages). A language is decidable but its complement is undecidable, or vice versa: a language is undecidable but its complement is decidable.

- *non-RE* - no TM can even recognize the members of the language in the RE sense (e.g., the diagonalization language). Neither a language nor its complement is decidable. Decidable problems have a (recursive) algorithm, i.e., TM halts whether or not it accepts its input. Decidable problems are described by recursive languages.

Classical algorithms as we know are associated with the class of recursive languages, a subset of recursively enumerable languages for which we can construct its accepting TM. For recursive languages, both a language and its complement are decidable. Turing machines are used primarily as a formal model of classical (recursive) algorithms.

An *algorithm* should consist of a finite number of steps, each having well defined and implementable meaning. We are convinced that computer computations are not restricted to such restrictive definition of algorithms only. If we allow for an infinite number of steps (e.g., reactive programs) and/or not well defined/implementable meaning of each step (e.g., an Oracle), we are in the class of super-recursive algorithms [5].

It is clear that Turing machines can compute recursive languages; they can compute REnon-rec languages (but not always decide whether a string belongs to the language, i.e., computation may not be effective in the sense of string acceptance/halting). By definition, Turing machines cannot represent non-RE languages. Computations that can do this are outside the Turing machine model. In other words, we cannot construct a TM able to accept strings exactly from a non-RE language. In this sense Turing machine is incomplete, and does not model all computations. Now we are ready to provide a more formalized definition of hypercomputation.

Definition 1. *By* superTuring computation models *(also called* superrecursive or hypercomputation models*), we will mean abstract devices able to compute and decide recursive languages, as well as to compute and decide recursively enumerable non-recursive REnon-rec languages and/or to compute (and perhaps decide) recursively non-enumerable non-RE (TM non-computable) languages.*

Such superTuring computation models are called super-recursive automata and rules that control their functioning are called super-recursive algorithms. The real question is whether TM non-computable languages and their corresponding computations can be expressed in alternative computational models? In the next section, we provide an affirmative answer to this question.

3 Interaction Machines Accepting TM Non-Computable Languages

We will present a model of interactive computation that will be able to express TM non-computable languages. We do not claim here that this model is the best and express all possible interactive computation.

Oue goal is to capture more precisely expressiveness and completeness of models of computation.

Definition 2. *We say that a class* **B** *of automata has* more accepting power *or is* more acceptably expressive *than a class* **A** *of automata if any language accepted by an automaton A from* **A** *is also accepted by some automaton B from* **B** *and there exist languages accepted by automata from* **B** *and not accepted by automata from* **A**.

For instance, Turing machines have more accepting power than pushdown automata, while pushdown automata have much more accepting power than finite automata.

It is possible to define many possible models of interactive computation, including Petri nets, neural networks, Gurevich's ASM [16], Persistent Turing Machines [15], grid automata [4], process algebras, such as CSP [19], CCS or π-calculus [20], ACP process algebra [2, 3], EAP algebra of processes [8, 9], Evolving Interactive Systems [25], and $-calculus [14]. However, we will use the model based on Interaction Machines, because of its simplicity and generality. Interaction Machines have been introduced by Wegner in 1997 to describe interaction of object-oriented and distributed systems [27, 28]. Interaction Machines are divided into two classes: Sequential Interaction Machines (SIMs) and Multi-Stream Interaction Machines (MIMs). SIMs are stream processing machines that model sequential interaction by I/O streams. MIMs are finite agents that interact with multiple autonomous streams. We extend here substantially the original Wegner's definitions.

In addition, we distinguish Cluster Interaction Machines (CIM) and Individual Interaction Machines (IIM). Each CIM consists of several interacting automata, e.g., Turing machines, while an IIM is a single automaton that interacts with its environment. Wegner [27, 28] introduced two types of Individual Interaction Machines: Sequential Individual Interaction Machines (SIIMs) and Multi-Stream Individual Interaction Machines (MIIMs). Inductive Turing machines [5] and Persistent TM [15] are examples of SIIMs. Internet machines [24] and grid automata that work with separate words [4] are examples of SCIMs. Here we introduce and study a model of interactive computations, which subsumes both MIMs and SIMs.

Let **K** be a class of abstract automata that work with (sets of) words in a finite alphabet X.

Definition 3. *A* linear interaction machine *(LIM) over the class* **K** *is a (possibly infinite) sequence* $E = \{A[i]; i = 0, 1, 2, 3, ...\}$ *of automata* $A[i]$ *from* **K** *where the first set of words* $X[0]$ *is given in advance and each automaton* $A[i]$ *is called a* component*, or* level automaton*, of E representing a one-level interaction algorithm that works with the input set of strings* $X[i]$ *and outputs the set of strings* $X[i + 1]$.

Note that IMs described above constitute MIMs in a general case, however, if each set $X[i]$ consists of one word, then we get SIMs.

Note that LIMs constitute only a special case of Interaction machines. They restrict the interaction to recognition of finite words with a single input at the very beginning of the computation, and they restrict interaction among the machines to simple message passing from one machine to the next one, in one direction. Intuitively, one would expect that interaction

will be a potentially an infinite process, with additional inputs from the environment during the computation. However, the goal of this paper is not to capture all possible forms of interaction, but rather to demostrate in a simple way that interaction captures computation that Turing machine cannot. For this purpose LIM is sufficient.

Of course, it is possible to define many other subclasses of Interaction machines. For example, it is possible to consider the following classes of interaction machines:

- the class **IIM** of all individual interaction machines;
- the class **SIIM** of all sequential individual interaction machines;
- the class **MIIM** of all multi-stream individual interaction machines;
- the class **SIM** of all sequential interaction machines;
- the class **MIM** of all multi-stream interaction machines;
- the class **CIM** of all cluster interaction machines;
- the class **SCIM** of all sequential cluster interaction machines;
- the class **MCIM** of all multi-stream cluster interaction machines;
- the class **LIM** of all linear interaction machines.

The original definitions of Wegner considered IIM (individual interaction machines) and two their special subclasses SIIM (sequential individual interaction machines) and MIIM (multi-stream interaction machines) [27, 28] and [12, 15, 16, 29]. IIM abstract from interaction channels assuming that interaction occurs between an individual interaction machine and an abstract environment. In reality, the environment may consist of other interaction machines or clusters of interaction machines combined by aribitrary interconnection topologies (bi- or uni-directional channels). Interaction machines IM can interact through single streams (SIM) or multiple streams (MIM). The components of interaction machines may constitute either atomic individual interaction machines IIM or clusters of interaction machines CIM (where each component is either atomic or another cluster). In particular, if interaction channels form a linear unidirectional chain topology, we receive LIM (i.e., linear interaction machines).

We leave other subclasses of interaction machines for future research and concentrate only on LIMs, because this is sufficient to the main goal of this paper, i.e., to prove that even a simple subclass of interaction machines like LIM is tremendously powerful and leads us outside of the Turing machine.

Adding communication channels allows to consider local and terminal behaviors of interaction machines that is crucial to understand the emerging behavior of IMs and their expressiveness.

Local behavior of the components from an IM $E = \{A[i]; i = 0, 1, 2, ...\}$ defines local languages accepted by the automaton E.

Definition 4. *A* local language *LL(E) of the automaton E is the language $L(A[i])$ accepted by one of the components of the automaton E, i.e., $A[i] \in E$, $i = 0, 1, 2, ...$*

Local behavior reflects non-coordinated functioning of the interaction machine components. Terminal behavior of the interaction machine E defines the *terminal language* accepted by this automaton.

Definition 5. *A word w is accepted in the* terminal mode *of the automaton E if given the word w as input to the automaton E, there is a number n such that the automaton $A[n]$ from E comes to an accepting state.*

Note that there are other useful modes of accepting a word by an automaton.

Definition 6. *The* terminal language *TL(E) of the automaton E is the set of all words accepted in the* terminal mode *of the automaton E.*

Local languages are special cases of terminal languages. Note that local languages are of the class of component automaton, i.e., regular, context-free, and so on. However, multiple interactive components can be more expressive than their individual components due to their interaction.

Theorem 1. *TL(***LIM***) coincides with class of all languages in the alphabet X.*
Proof: *We show that given a formal language L, i.e., a set of finite words in the alphabet X, there is an SIM A such that A accepts L. We assume that all components of $A = \{A[t]; t = 0, 1, 2, 3, \ldots\}$ work in sequential order corresponding to their indices in generations $t = 0, 1, 2, 3, \ldots$. Each component of SIM takes as input a string and produces as output a string to pass it to the successor (if it does not accept) or stops the computation if the string is accepted. To do this, for each word w, we build a finite automaton A_w that given a word w as its input, it accepts only the word w, and given any other word u, it outputs u, which goes as input to the next component in the SIM A. In such a way a component interacts with its predecessor and its successor. In both cases, the automaton A_w comes to a terminal state. Then taking any sequence $E = \{A[t] = A_w; w \in L\}$ of such automata, we obtain the necessary sequential interaction machine A.*

Remark 1. Note that Interaction Machines in the above proof do not gain their computational power by interacting with more powerful systems nor by accessing noncomputable information (like is done in TM with an oracle). The interacting components are very simple (of the class of finite automata) and components of distributed information are also very simple (single strings, i.e., finite subsets of regular languages) − however, their combination (interaction/agglomeration) makes the power. The situation will not change if we replace nodes by Turing machines − of course, each node can do more, but together they cannot do more than interactive finite automata (it is possible to suggest that this was intuitively well known to von Neumann and Ulam when they proposed cellular automata as a model of universal constructability [26]). The real advantage is the following: we have only conventional interacting TMs (finite automata) and it is possible to compute more than a single non-interacting TM can.

Remark 2. Note that Interaction Machines allow us to decide arbitrary sets of strings in the alphabet X. This includes any recursive language, any recursively enumerable but not recursive language, and any non-recursively enumerable language. In this sense, interaction machines are more complete than Turing machines able to model and decide recursive languages, and model recursively enumerable languages only. Of course, to apply the above theorem in practice, we have to know all the strings belonging for example to the diagonalization language - a standard representative of non-RE class. To do that we have to find firstly the characteristic vector for every Turing machine in the diagonalization table, i.e., to solve the TM halting problem. For that we need an enumerable (infinite) number of generations. Thus the proof is not effectively constructive but existential. Nevertheless, it can represent and decide an arbitrary language from any of three classes and can be simulated in interpreted dynamic languages like Ruby in the style of reactive programs.

In [7], we proved an analogous result for the class of *Evolutionary Finite Automata* (**EFA**) showing that not only interaction but evolution is more expressive than Turing machines. In this sense interaction and evolution are linguistically similar.

Remark 3. The problem of finding sufficient and necessary conditions for achieving superTuring power by interaction of a finite number of ordinary automata, such as finite automata, Turing machines or RAM, is completely solved in [6].

4 Questions about Completeness of Interaction Machines and Hypercomputation

As the consequence of Theorem 1 we get

Corollary 1. *Interaction Machines working in terminal mode allow to model all recursive, REnon-rec, and non-RE languages, i.e., more than Turing machines can express.*

Remark 4. All these results give an impression that Interaction machines, one of hypercomputation models, are *computationally complete*. However, theoretical intuition seems to suggest that all hypercomputational models will be more complete, but never absolutely complete. Is this a paradox?

To correctly understand this situation, we must make clear that the problem is associated with the meaning of the term completeness. If we assume that three classes - recursive, REnon-rec, non-RE languages - provide a complete classification of languages, then the IM model would be complete. However, the above classification is essentially incomplete and imprecise. For example, the class of recursive languages consists of many subclasses (polynomial, exponential, NP-complete, context-free, if to list just a few, and scientists invent continuously new subclasses). The class nonRE has more representatives of languages than the two other classes combined together (the number on nonRE languages corresponds to the number of real numbers versus enumerable integers/RE languages) and also consists of many subclasses. Some of them are described by the arithmetical hierarchy and by the inductive hierarchy. In addition, we encounter a similar situation with the class of languages computable in the limit.

Besides, there are automata that perform computations with infinite words, as well as with non-linear structures, such as graphs.

All these considerations show that we do not have a complete model for computation (a lot of details are missing). It is possible to conjecture that a complete theory of computation cannot be created but it may and should be approximated (we can call it, somehow jokingly from the names of the authors, the WEB thesis and suggest to see how fast and successfully will be challenged).

5 Conclusion

In this paper, we demonstrated that Turing machines do not model all computations and thus, do not provide means for a theory of everything in computer science. We presented and proved that Interaction machines are more complete than Turing machines. However, we do not claim that Interaction machines are computationally complete. We believe that in fact the search for complete theory of computation might be infinite.

Besides, as we know, no argument against the existence of such a theory of everything has gained general acceptance. Whether the next level of computer science theory give us the actual

theory of everything is unknown and this is the reason that scientists will attempt to find a next good candidate for theory of everything. We believe that this is probably not possible.

In the closing statement to the Ubiquity symposium on computation Peter Denning wrote about an emerging consensus that reactive systems are models of interactive computation and are different from Turing modeled systems [11]; and Alfred Aho advocated for new models of computation: "as the computer systems we wish to build become more complex and as we apply computer science abstractions to new domains, we discover that we do not always have the appropriate models to devise solutions. In these cases, computational thinking becomes a research activity that includes inventing appropriate new models of computation" [1]. The above is consistent with our results.

References

[1] A. Aho. *Computation and Computational Thinking* Ubiquity Symposium, ACM, 2010, `http://ubiquity.acm.org/symposia.cfm`

[2] J.C.M Baeten and J.A. Bergstra. *Discrete time process algebra.* Formal Aspects of Computing, 1996, v. 8, no. 2, pp. 188-208.

[3] J.A. Bergstra and J.W. Klop. *Process algebra for synchronous communication.* Information and Control, 1984, Vol. 60, Issue 1-3, pp. 109-137.

[4] M. Burgin. *Cluster Computers and Grid Automata.* in Proceedings of the ISCA 17th Intern. Conf. "Computers and their Applications", Intern. Society for Computers and their Applications, Honolulu, Hawaii, 2003, pp. 106-109.

[5] M. Burgin. *Superrecursive Algorithms.* Springer, New York, 2005.

[6] M. Burgin. *Interactive Hypercomputation.* in Proceedings of the 2007 International Conference on Foundations of Computer Science (FCS'07), CSREA Press, Las Vegas, Nevada, USA, 2007, pp.328-333.

[7] M. Burgin and E. Eberbach. *Evolutionary Automata: Expressiveness and Convergence of Evolutionary Computation.* Computer Journal, 2011 (doi:10.1093/comjnl/bxr099).

[8] M. Burgin and M.L. Smith. *A Unifying Model of Concurrent Processes.* In Proceedings of the 2007 Intern. Conf. on Foundations of Computer Science (FCS'07), (H.R. Arabnia and M. Burgin, Eds.) CSREA Press, Las Vegas, Nevada, USA, 2007, pp. 321-327.

[9] M. Burgin and M.L. Smith. *A Theoretical Model for Grid, Cluster and Internet Computing.* in Selected Topics in Communication Networks and Distributed Systems, World Scientific, New York/London/Singapore, 2010, pp. 485-535.

[10] A. Church. *An Unsolvable Problem of Elementary Number Theory.* American Journal of Mathematics, 1936, vol.58, 345-363.

[11] P. Denning. *What Have We Said About Computation?* Closing Statement, ACM, Ubiquity Symposium, 2011, `http://ubiquity.acm.org/symposia.cfm`

[12] E. Eberbach and P. Wegner. *Beyond Turing Machines.* Bulletin of the European Association for Theoretical Computer Science, 2003, 81, 279-304.

[13] E. Eberbach, D. Goldin and P. Wegner. *Turings Ideas and Models of Computation.* in: (ed.Ch.Teuscher) Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, 2004, 159-194.

[14] E. Eberbach. *The $-Calculus Process Algebra For Problem Solving: A Paradigmatic Shift in Handling Hard Computational Problems.* Theoretical Computer Science, 2007, vol.383, no.2-3, 200-243.

[15] D. Goldin. *Persistent Turing Machines as a Model of Interactive Computation.* in Found. of Information and Knowledge Systems, LNCS 1762, Springer-Verlag, Berlin, 2000, pp.116-135.

[16] D. Goldin, S. Smolka and P. Wegner. (eds.) *Interactive Computation: the New Paradigm.* Springer-Verlag, 2006.

[17] K. Gödel. *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme.* Monatschefte für Mathematik und Physik, 1931, 39, 173-198.

[18] S. Hawking. *The Theory of Everything: The Origin and Fate of the Universe.* 2002.

[19] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall International Series in Computer Science, Prentice-Hall International, UK, 1985.

[20] R. Milner, J. Parrow and D. Walker. *A Calculus of Mobile Processes, I & II.* Information and Computation, 1992, 100, 1-77.

[21] A. Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem.* Proc. London Math. Soc., 42-2, 1936, 230-265; A correction, ibid, 43, 1937, 544-546.

[22] A. Turing. *Systems of Logic based on Ordinals.* Proc. London Math. Soc. Series 2, 1939, 45, 161-228.

[23] Turing, A. Intelligent Machinery, 1948, in Collected Works of A.M. Turing: Mechanical Intelligence, ed.D.C.Ince, Elsevier Science, 1992.

[24] J. van Leeuwen and J. Wiedermann. *The Turing Machine Paradigm in Contemporary Computing.* in. B. Enquist and W. Schmidt (eds.) Mathematics Unlimited - 2001 and Beyond, LNCS, Springer-Verlag, 2001.

[25] P. Verbaan, J. van Leeuwen and J. Wiedermann. *Complexity of Evolving Interactive Systems.* Springer-Verlag, LNCS 3113, 2004, 268-281.

[26] J. von Neumann. *Theory of Self-Reproducing Automata.* (edited and completed by Burks A.W.), Univ. of Illinois Press, 1966.

[27] P. Wegner. *Why Interaction is More Powerful Than Algorithms.* CACM, 1997, vol.40, no.5, 81-91.

[28] P. Wegner. *Interactive Foundations of Computing.* Theoretical Computer Science, 1998, 192, 315-351.

[29] P. Wegner and E. Eberbach. *New Models of Computation.* Computer Journal, 2004, vol.47, no.1, 4-9.