New Models of Computation

PETER WEGNER[1] AND EUGENE EBERBACH[2]

[1]*Department of Computer Science, Brown University, Providence, RI 02912, USA*
[2]*Computer and Information Science Department, University of Massachusetts, North Dartmouth, MA 02747, USA*
Email: pw@cs.brown.edu, eeberbach@umassd.edu

This paper examines the limitations of Turing Machines as a complete model of computation, and presents several models that extend Turing Machines. Dynamic interaction of clients and servers on the Internet, an infinite adaptation from evolutionary computation, and robots sensing and acting are some examples of areas that cannot be properly described using Turing Machines and algorithms. They require new models of computation going beyond Turing Machines. We refer to such new models as superTuring models of computation. Three superTuring models of computation, namely Interaction Machines, the π calculus and the $-calculus are presented and explained, focussing on why they are better for the solution of computational problems. We expect that superTuring computation will become the central programming paradigm in the future.

Received 21 June 2002; revised 17 May 2003

1. INTRODUCTION

This paper examines the limitations of Turing Machines (TMs) as a complete model of computation, and presents several models that extend Turing Machines.

Alan Turing's 1936 paper 'On Computable Numbers, with an Application to the Entscheidungsproblem' [1] showed that Hilbert's assumption of logical provability of all mathematical theorems [2] could not be proved by computers and algorithms. Turing showed that computers were too weak to prove all mathematical decision problems and by implication too weak to solve all mathematical decision problems. He introduced choice machines and oracle machines [1, 3] more expressive than Turing Machines in their problem solving ability. But Turing Machines were used later as a paradigmatic model of computation, and the Church–Turing thesis was adopted in the 1960s as a complete model for algorithms and general problem solving. The belief that Turing Machines provided a universal model for problem solving was inconsistent with Turing's proof that they could not solve all mathematical problems, and by implication all computational problems. Turing's proof that the Entscheidungsproblem is not solvable implies that not all computational problems are solvable algorithmically by computers.

The unsolvability of the Entscheidungsproblem was first proved by Gödel using recursive functions [4], and later by Church using λ-calculus [5]. All three methods turned out to be equivalent. The equivalence of three classes of functions (recursive, lambda-definable and Turing-computable) was taken as confirmation that the notion of effective function computation has been formally properly defined. This led in the 1950s to their combination into the Church–Turing thesis:

> The equivalent notions of recursiveness, lambda-definability and Turing-computability capture the notion of effective computability of functions over integers.

The simplicity, elegance and power of Turing Machines became a paradigm of computation that destroyed earlier assumptions that all mathematical and computational problems could not be algorithmically solved. In the 1960s, with the development of a new ACM computer science curriculum based on algorithms, Turing Machines were chosen to fulfill a central role in computer science. Additionally, when the first undergraduate computer science textbooks were being written in the 1960s, the Church–Turing thesis was extended incorrectly to its strong form:

> Whenever there is an effective method for doing something with a computer, it can be computed by a Turing Machine.

However, the original Church–Turing thesis only applies to the effectiveness of computing functions over integers, and it does not extend to other types of computation, such as functions over reals or such as interactive computation. Both Church and Turing were aware of this limitation, and Turing proposed other, more powerful, models of computation. However, all limitations of algorithmic computation as embodied in the Turing Machine to the exclusion of other types of computation, were later ignored. While computation beyond Turing Machines was of interest to Turing himself, it is not consistent with the theory of computation that developed in the 1960s [6], and whose cornerstones were Turing Machines and algorithms.

With the development of new powerful applications, it is becoming evident to the wider computer science community that Turing Machines and algorithms do not provide a complete model for problem solving. Dynamic interaction of clients and servers on the Internet, an infinite adaptation from evolutionary computation, and robots sensing and acting in complex environments, are some examples of areas which cannot be described properly using Turing Machines and

algorithms. They require new models of computation going beyond Turing Machines. We refer to such new models, able to carry out non-algorithmic computation, as superTuring models.

In this paper we present three superTuring models, namely, Interaction Machines (IMs), the π-calculus and the \$-calculus, and explain why they are better for the solution of complex computational problems than Turing Machines. We expect that superTuring models of computation will become an active area of research for many years to come. This paper encourages more research on new models of computation for problem solving.

2. TURING MACHINES AND ALGORITHMS

Turing Machines and algorithms are two fundamental concepts of computer science and problem solving. Turing Machines describe the limits of algorithmic problem solving, and laid the foundation of current computer science in the 1960s.

The *TM* is considered a formal model of a (digital) computer running a particular program. The Turing Machine is the invention of Alan Turing, who introduced his automatic machine (a-machine) in his 1936 paper [1] as a byproduct to show the unsolvability of the Hilbert decision problem in mathematics to prove or disprove all mathematical statements [2]. The TM is supposed to model an arbitrary problem solving using mechanical methods.

It turns out that *undecidable problems* cannot be solved by TMs and *intractable problems* are solvable, but require too many steps. For undecidable problems effective recipes do not exist—problems are called non-algorithmic or non-recursive. 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.

The number of different languages (problems) over any alphabet of more than one symbol (at least two symbols in the alphabet are needed: one to encode the words and the second to separate them) is not countable; however, the number of all possible TMs is enumerable. Each TM represents one problem (language). Thus it is clear that there exist problems which cannot be solved by TMs. In such a way Turing received his result about unsolvability of the 'halting problem' of the Universal Turing Machine representing all possible TMs. This corresponds directly to the unsolvability of Hilbert's *Entscheidungsproblem*—the main reason for introduction by Turing of his a-machines.

An *algorithm* is one of the most fundamental concepts of current mathematics and computer science having roots in ancient Egypt, Greece and Babylon, introduced with the attempt to 'mechanize' computational problem solving. The algorithm should consist of a finite number of steps, each having well-defined meaning. The general belief that every algorithm can be expressed in terms of a Turing machine, is now known as the *Church–Turing thesis*. The simplicity of the TM model is used to prove formally that there are specific problems (languages) that the TM cannot solve.

The TM model is too weak to describe properly the Internet, evolution or robotics because it is a closed model, which requires that all inputs are given in advance, and TM is allowed to use an unbounded but only finite amount of time or memory resources [7, 8].

In the case of the Internet, the web clients 'jump' into the middle of interaction, without a knowledge of the server state and previous history of other clients. A dynamic set of inputs and outputs, parallel interaction of multiple clients and servers, a dynamic structure of the Internet communication links and nodes, is outside what a sequential, static and requiring full specification Turing Machine can represent.

TMs also have problems with capturing evolution, because solutions and evolutionary algorithms are changed in each generation, and the search for a solution (represented by a global optimum) is in general an infinite process. The above is true independently whether search spaces are finite or infinite. This is so because the search for a solution in evolutionary computation is typically probabilistic, and then the infinite number of steps may be required for finite search spaces as well.

Robots interact with environments which are very often more complex than robots themselves, and even worse—the environments can be non-computable. Thus it is clear why conventional computer science—Good Old Fashioned AI (GOFAI) failed miserably, when it tried to build a model of a robot and its environment using Turing Machines and algorithms (i.e. deliberative goal-based robots building the model of the world to plan their actions, versus behavior-based robots reacting only to their sensory inputs, because algorithmic planning did not work).

3. BEYOND TURING MACHINES: THREE SUPERTURING MODELS OF COMPUTATION

Theoretical computer science has now undergone several decades of development. The 'classical' topics of automata theory, formal languages and computational complexity have become firmly established, and their importance to other theoretical work and to practice is widely recognized. However, in many areas like multi-agent interactive systems, web-distributed programming, robotics, reactive systems, client–server models, distributed databases, it turned out that algorithms do not describe well their behavior. The biggest challenge seems to be the development and acceptance of computational models more expressive than Turing Machines, referred here as superTuring models of computation.

> By *superTuring computation* we mean any computation that cannot be carried out by a Turing Machine as well as any (algorithmic) computation carried out by a Turing Machine.

In this paper, we will use the term computation in this wider 'superTuring computation' sense, and not in the narrower 'algorithmic' meaning.

> We define a *superTuring computer* as any system or device capable of carrying out superTuring computation.

It looked that after several decades of trying to find a stronger model of computation, a majority of computer scientists gave up and commonly accepted the Church–Turing thesis that every algorithm can be described by (or implemented on, or is equivalent to) a Turing Machine. There is no problem with equating classical algorithms and Turing Machines. The problem is with equating all forms of computation (i.e. including superTuring computation) with algorithms and Turing Machines. Algorithms are elegant and useful, and it would be nice if it was possible to describe all computations by algorithms. However, this is not possible.

In this section we present three superTuring models of computation that are more expressive than Turing Machines and algorithms: Interaction Machines, the π-calculus and the \$-calculus. We discuss why they are better than TMs for the solutions of computational problems (and, in particular, for handling three troublesome areas mentioned before: the Internet, evolution and robotics). All three models capture directly the nature of open systems interacting with an undetermined environment.

3.1. Interaction Machines

Interaction Machines have been introduced by Wegner in 1997 to describe interaction of object-oriented and distributed systems [9, 10].

The paradigm shift from Algorithms to Interaction captures the technology shift from mainframes to workstations and networks, from number crunching to embedded systems and user interfaces, and from procedure-oriented to object-based and distributed programming. Turing Machines are too weak to express interaction of object-oriented and distributed systems, and Interaction Machines are proposed as a stronger model that better captures computational behavior for finite interactive computing agents. The intuition that computing corresponds to formal computability by Turing Machines breaks down when the notion of what is computable is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing Machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid [10].

Interaction Machines extend the Turing Machine model by allowing *interaction*, i.e. input and output actions (read and write statements) determined by the environment at each step of computation. This extends the closed TM model to an open system. Whereas Turing Machines require all inputs to appear on the tape prior to the computation and shut out the world during the process of computation, Interaction Machines allow inputs to be generated dynamically and require inputs to be represented by a potentially infinite stream, since any finite stream can be dynamically extended by interactive inputs. Interactive systems interact with an external environment they cannot control. Interactive components can always extend any finite sequence. The behavior of interactive systems is better modeled by infinite processes that express the cardinality of the real numbers (Cantor diagonalization) than by enumerable sequences. Interaction Machines correspond to Turing Machines with unbounded tape contents. Whereas the number of inputs of a Turing Machine is countable, the number of potential streams of an Interaction Machine is non-enumerable. However, IMs, dissimilar to TMs, use *interaction* as a basic principle of their study that is a foreign notion for the closed world of TMs. Whereas the TM 'knows' in advance the contents of its tape, the IM never is sure which input it will receive as the result of interaction. The evolution (or dynamicity) of bounded inputs and outputs makes IMs different from TMs.

The expressiveness of Interaction Machines is due to unbounded tape contents, enumerable (potentially infinite) set of states, inputs and outputs. The set of inputs is dynamically bounded (it evolves). Additionally, it is assumed that other agents (environment to interact with) can be non-computable, which leads to a more expressive model than TM.

Interaction Machines can be used to model the dynamic interaction on the Internet, because they allow for dynamically bounded inputs (i.e. they may be unknown for the clients until the last moment), the number of states, inputs and outputs can be infinite.

The canonical model of Interaction Machines, so-called Persistent Turing Machines [11], are suitable for evolutionary computation. Persistent Turing Machines preserve the contents of its tape from computation to computation, which allows to model evolution preserving gene information from generation to generation.

Sensing and acting in robotics is much more natural to model in Interaction Machines than in Turing Machines. Dynamically bounded sensory inputs allow to express both uncertainty of sensors and actuators.

3.2. The π-calculus

Milner's π-calculus [12, 13, 14] is a mathematical model of processes whose interconnections change as they interact. Similar to Interaction Machines, the π-calculus is built around the central notion of *interaction*. It is a model of the changing connectivity of *interactive systems*. In such a sense the π-calculus of mobile processes can be considered to be a dynamic extension of Milner's earlier model—Calculus of Communicating Systems (CCS) [15]. Like the majority of process algebras, the π-calculus uses a synchronous message passing by handshaking. Typically process algebras have static communication channels. The π-calculus allows an arbitrary link topology and to change it by sending new links through existing ones. The ability to directly represent mobility, in the sense of processes that re-configure their interconnection structure when they execute, makes it easy to model systems where processes move between different locations and where resources are allocated dynamically.

The π-calculus can be seen as a basic model of computation both in the 'narrower' algorithmic and wider 'super-Turing' sense [12, 13, 14]. Every basic model rests upon a small number of primitive notions; the π-calculus rests upon the primitive notion of *interaction*, just as Turing Machines rest upon the notion of reading and writing a storage medium, and just as recursive equations and the λ-calculus rest upon mathematical function reduction. The π-calculus

subsumes the canonical approach to sequential computing; i.e. Church's λ-calculus [16]—the above means that the π-calculus is at least equally expressive to Turing Machines.

According to [14] there is yet no definite measure on expressiveness of the π-calculus, although there is strong evidence that its primitives are enough for a wide variety of purposes, including encoding of the functional and object-oriented paradigms.

We justify below that the π-calculus might be more expressive than TMs. Everything depends on the interpretation of the π-calculus replication (or its predecessor: recursive definition) operator. The π-calculus replication operator of the process expression is defined as the parallel composition (iteration) of the process as many times as needed. If the replication is unbounded (infinite) then the π-calculus becomes more expressive than TMs. In particular, then it can model an infinite number of cells of cellular automata, discrete neural networks or random automata networks, i.e. models more expressive than TMs (see, e.g. [17]). Any cellular automaton or neural network could be modeled in the π-calculus as an infinite parallel composition of corresponding cells or neurons respectively. Cells or neurons can interact by the π-calculus message-passing in and out operators. Without infinity, unless to assume that the π-calculus processes have the power of oracles, we remain in the class of TMs. Our interpretation is that *as many times as needed* implies the infinite number of replications. With such interpretation of unbounded replication (or recursive definition, or parallel composition), the π-calculus is more expressive than TMs.

The π-calculus provides the following support for interaction on Internet, evolution and robotics. In the π-calculus, we can model in a natural way interaction on the Internet by message-passing. Communication topology can be changeable, because the π-calculus allows for link mobility. It is possible to model both asynchronous and synchronous send and receive, and point-to-point communication can be used (although not very economically) to express unicasting, multicasting and broadcasting.

For evolutionary computation, the π-calculus, because of its inherent parallelism, allows in a natural way to express a population of solutions or co-evolution. Otherwise no more support is provided.

Robots may interact by the π-calculus send (out) and receive (in) communication primitives through dynamically created communication channels.

3.3. The $-calculus

The $-calculus is a mathematical model of processes capturing both the final outcome of problem solving as well as the interactive incremental way how the problems are solved. The $-calculus is a process algebra of Bounded Rational Agents for Interactive Problem Solving. It has been introduced in the late 1990s [18, 19, 20]. The $-calculus (pronounced cost calculus) is a formalization of resource-bounded computation (called also anytime algorithms, proposed by Dean, Horvitz, Zilberstein and Russell in late 1980s and beginning of the 1990s [21]). Anytime algorithms are guaranteed to produce better results if more resources (e.g. time, memory) become available.

The $-calculus rests upon the primitive notion of *cost* in a similar way as the π-calculus was built around a central concept of *interaction*. Cost and interaction concepts are interrelated in the sense that cost captures the quality of an agent interaction with its environment. The unique feature of the $-calculus is that it provides a support for problem solving by incrementally searching for solutions and using cost to direct its search. The basic $-calculus search method used for problem solving is called $k\Omega$-optimization. It is a very general search method, allowing to simulate many other search algorithms, including A*, Minimax, dynamic programming, tabu search or evolutionary algorithms. The $-calculus is applicable to robotics, software agents, neural nets and evolutionary computation. Potentially it could be used for design of cost languages, cellular evolvable cost-driven hardware, DNA-based computing and molecular biology, electronic commerce and quantum computing.

The $-calculus derives its expressiveness from the *infinity* of its operators rather than from the 'magic' of oracles [3]. It is easier and 'cleaner' to think about implementation of unbounded (infinite) concepts, than about implementation of oracles. The implementation of scalable computers (e.g. scalable massively parallel computers or unbounded growth of the Internet) allows to think about a reasonable approximation of the implementation of *infinity* (and, in particular, Interaction Machines, the π-calculus or the $-calculus). At this point, it is completely unclear how to implement oracles (as Turing stated *an oracle cannot be a machine* [3], i.e. implementable by mechanical means), and as the result, the models based on them.

The $-calculus has the same features to model the Internet client–server interaction as the π-calculus, i.e. communication by message-passing, and the ability to create and destroy processes and communication links. On top of that, by using its cost mechanism, it allows to model allocation of resources, to optimize flow of information or quality of service. Because of infinity of its operators (i.e. it permits for an infinite sequence of actions, or unbounded parallelism), there is no problem with scalability.

The $-calculus allows in a natural way to express evolution. Its cost performance measure allows to model fitness function, and an infinite search for global optima. Additionally, $-calculus allows both to optimize the quality of solutions and to minimize search (problem solving) cost.

Behavior of robots, and their interaction with an environment and other robots can be expressed in a uniform form by the $-calculus-based Generic Behavior Message-Passing Language (GBML) [18]. Uncertainty can be modeled by combining choice operator with costs using, for instance, probabilities or fuzzy sets membership function.

4. CONCLUSIONS

In 1968, with the development of a new ACM computer science curriculum based on algorithms [6], Turing Machines

were chosen to fulfill a central role in computer science, because of their elegance and simplicity, and for many years they have been useful in that function.

However, we used to forget that Turing was more concerned that the *Entscheidungsproblem* is unsolvable, and mathematics cannot be fully described by algorithms, rather than to propose or create the foundations of computer science. Turing himself did not accept the Turing Machine as a complete model for problem solving, and its acceptance by theoreticians contradicted Turing's view on this subject.

Turing Machines and algorithms are simple but incomplete. Theoreticians still think about TMs as a paradigm for a complete model of computation, and react negatively against any attempt to go beyond TMs. The reasons why this is happening are understandable. Practically most of the models developed so far turned out to be equally expressive to TMs. Additionally, very rarely in practical approaches, Turing Machines were stretched to the limits of their computational power in the attempt to solve TM undecidable problems. New applications and demands for solutions of a new class of problems have been changing that situation. This is consistent with the opinion of Turing himself.

Computer scientists quite easily forget that Turing has had very serious doubts about completeness of his model. Turing besides his commonly known TMs, introduced three other models of computation: choice machines [1], oracle machines [3] and unorganized machines [22], which he considered to be more expressive than his classical model. Also Von Neumann and Ulam proposed the solution of universal constructability using the help of more powerful cellular automata [23]. In the 1990s several other models more expressive than TMs have been proposed. They include, besides models discussed in Section 3, Persistent Turing Machines [11], Site and Internet Machines [24], discrete neural networks and automata networks [17], analog computation and real-value neural networks [25], Evolutionary Turing Machines [26], Inductive Turing Machines [27] and Accelerating Turing Machines [28]. The reader can find a more complete discussion of superTuring models in [7, 20], where also three basic principles underlying and unifying superTuring computation (interaction with the world, infinity of resources and evolution of systems) have been presented.

In this paper we discussed three superTuring models: Interaction Machines, the π-calculus and the \$-calculus. All three models are more powerful but also more complex than TMs. All three were designed to capture dynamic aspects of interactive computation, but designed for different purposes. Interaction Machines were designed to model dynamic aspects of computation based on interaction. The π-calculus also was developed around the central notion of interaction, but its primary objectives were to describe concurrent systems with changeable communication topology. On the other hand, the \$-calculus uses costs to capture adaptive problem solving searching for the best solutions under bounded resources.

We hope that this research will lead to a simple and more complete model of computation. We are working towards these objectives, and we hope that this paper will provide a framework to develop models going beyond TMs and addressing better current and future needs of computer science. Just as Turing Machines laid the foundations of theoretical computer science and of programming languages and architectures, the introduction (and acceptance) of superTuring models of computation will lead to new foundations, hardware and software for computers.

ACKNOWLEDGEMENTS

The authors would like to thank anonymous reviewers of *The Computer Journal* for their suggestions and comments.

REFERENCES

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

[2] Hilbert, D. and Ackerman, W. (1928) *Grundzüge der Theoretischen Logik*. Springer-Verlag, Berlin.

[3] Turing, A. (1939) Systems of logic based on ordinals. *Proc. London Math. Soc. Ser. 2*, **45**, 161–228.

[4] Gödel, K. (1931) Über formal unedtscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatschefte für Mathematik und Physik*, **38**, 173–198.

[5] Church, A. (1936) An unsolvable problem of elementary number theory. *Am. J. Mathematics*, **58**, 345–363.

[6] ACM Curriculum Committee (1968) Curriculum 68: Recommendations for academic programs in computer science. A Report of the ACM Curriculum Committee on Computer Science. *Commun. ACM*, **11**, 151–169.

[7] Eberbach, E., Goldin, D. and Wegner, P. (2003) Turing's ideas and models of computation. In Teuscher, Ch. (ed.), *Alan Turing: Life and Legacy of a Great Thinker*. Springer-Verlag, Berlin, Heidelberg, New York.

[8] Wegner, P. and Goldin, D. (2003) Computation beyond Turing Machines: seeking appropriate methods to model computing and human thought. *CACM*, **46**(4), 100–102.

[9] Wegner, P. (1997) Why interaction is more powerful than algorithms. *CACM*, **40**(5), 81–91.

[10] Wegner, P. (1998) Interactive foundations of computing. *Theor. Comp. Sci.*, **192**, 315–351.

[11] Goldin, D., Smolka, S. and Wegner, P. (2001) Turing Machines, Transition Systems, and Interaction. In *Proc. 8th Int. Workshop on Expressiveness in Concurrency*, August 2001. Aalborg, Denmark.

[12] Milner, R., Parrow, J. and Walker, D. (1992) A calculus of mobile processes, I & II. *Information and Computation*, **100**, 1–77.

[13] Milner, R. (1993) Elements of interaction. *CACM*, **36**(1), 78–89.

[14] Plotkin, G., Stirling, C. and Tofte, M. (eds) (2000) *Proof, Language, and Interaction: Essays in Honour of Robin Milner*. The MIT Press, Cambridge and London.

[15] Milner, R. (1980) *A Calculus of Communicating Systems, LNCS 94*. Springer-Verlag, Berlin, Heidelberg, New York.

[16] Church, A. (1941) *The Calculi of Lambda Conversion*, Princeton University Press.

[17] Garzon, M. (1995) *Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks, An EATCS series*. Springer-Verlag, Berlin, Heidelberg, New York.

[18] Eberbach, E., Brooks, R. and Phoha, S. (1999) Flexible Optimization and Evolution of Underwater Autonomous Agents. In Zhong, N., Skowron, A. and Ohsuga, S. (eds), New Directions in Rough Sets, Data Mining, and Granular-Soft Computing, *Proc. 7th Int. Workshop on Rough Sets, Fuzzy Sets, Data Mining and Granular-Soft Computing RSFDGrC'99*, Yamaguchi, Japan, LNAI 1711, pp. 519–527. Springer-Verlag, Berlin, Heidelberg, New York.

[19] Eberbach, E. (2001) $-Calculus bounded rationality = process algebra + anytime algorithms. In Misra J. C. (ed.), *Applicable Mathematics: Its Perspectives and Challenges*, Chapter 22, pp. 213–220. Narosa Publishing House, New Delhi, Mumbai, Calcutta.

[20] Eberbach, E. (2003) Is Entscheidungsproblem solvable? Beyond undecidability of Turing Machines and its consequence for computer science and mathematics. In Misra, J. C. (ed.), *Computational Mathematics, Modelling and Algorithms*, Chapter 1, pp. 1–32. Narosa Publishing House, New Delhi, Mumbai, Calcutta.

[21] Horvitz, E. and Zilberstein, S. (eds) (2001) Computational tradeoffs under bounded resources. *Artificial Intelligence*, **126**, 1–196.

[22] Turing, A. (1992) Intelligent machinery. In Ince, D. C. (ed.), *Collected Works of A.M. Turing: Mechanical Intelligence*. Elsevier Science.

[23] Von Neumann, J. (1966) *Theory of Self-Reproducing Automata*, (edited and completed by Burks, A. W.). University of Illinois Press, Urbana and London.

[24] Van Leeuwen, J. and Wiedermann, J. (2000) The Turing Machine paradigm in contemporary computing. In Enquist, B. and Schmidt, W. (eds), *Mathematics Unlimited—2001 and Beyond, LNCS*. Springer-Verlag.

[25] Siegelmann, H. (1999) *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, Boston, Basel, Berlin.

[26] Eberbach, E. (2002) On expressiveness of evolutionary computation: is EC algorithmic? In *Proc. 2002 World Congress on Computational Intelligence (WCCI)*, Honolulu, HI, May 12–17, 2002, pp. 564–569.

[27] Burgin, M. (2001) How we know what technology can do. *CACM*, **44**(11), 83–88.

[28] Copeland, B. J. (1998) Super-Turing machines. *Complexity*, **4**(1), 30–32.