Proceedings of the High Performance Computing Symposium,
***San Diego*, 1999, pp. 224-228**

SUPER-RECURSIVE ALGORITHMS AS A TOOL FOR HIGH PERFORMANCE COMPUTING

M.Burgin
Department of Mathematics
University of California, Los Angeles
Los Angeles, CA 90095

ABSTRACT: *All approaches to high performance computing is naturally divided into three main directions: development of computational elements and their networks, advancement of computational methods and procedures, and evolution of the computed structures. In the paper the second direction is developed in the context of the theory of super-recursive algorithms. It is demonstrated that such super-recursive algorithms as inductive Turing machines are more adequate for simulating many processes, have much more computing power, and are more efficient than recursive algorithms.*

1. INTRODUCTION

The variety of approaches to high performance computing is naturally divided into three main directions. The first one is aimed mostly at the development of computational elements and their networks. It is the *hardware oriented* approach. The second direction is focused on the advancement of computational methods and procedures. It is the *software oriented* approach. The third direction is pointed at the evolution of the computed structures (such as data and knowledge) and enhancement of the processed information. It is the *infware oriented* approach.

Our main topic here is related to the second direction. However, to elaborate more efficient software in practice, it is necessary to have sufficiently powerful hardware. Consequently, some issues of the first direction are also treated here in connection with the problems of computational methods and procedures under discussion.

As it has been demonstrated by the history of computer science and computing practice, development of the second direction can go on by enhancing and enriching the existing methods and procedures, but only inventing extremely original computational structures usually performs the real breakthrough.

In the theory of algorithms such a step has been made by transition from the ordinary, recursive algorithms (such as Turing machines, partial recursive functions, Minsky machines, random access machines (RAM), Kolmogorov algorithms etc.) to the super-recursive algorithms violating the famous Turing-Church thesis (Burgin 1987).

2. RECURSIVE AND SUPER-RECURSIVE ALGORITHMS

Development of the theory of algorithms inspired different authors to suggest new mathematical models for a general definition of algorithm. However, in all cases considered as algorithmic, it was proved that all mathematical models of algorithms were functionally equivalent. It caused introduction of the famous Turing-Church thesis (conjecture). It claims that *any mathematical model of an algorithm is functionally equivalent to a Turing machine*. For many years, attempts to find more powerful models than Turing machines gave no positive results.

The traditional formalism for a substantive concept of algorithm in general presupposes three essential conditions:

1) finiteness and constructibility of the input, processed, and output information over a finite interval of time;
2) finiteness and constructibility of the system of rules for realization of the processing;
3) purposefulness of the system of rules. In this context constructibility means that it is possible to generate and transform data as well as to realize the rules by means of a simple mechanical device.

In contrast to the informal notion of algorithm, other conditions of algorithm specification appear only in mathematical models of algorithms. One of these extra conditions is the demand that to obtain a result of a computation, it is necessary to finish the computational process. However, it is superfluous in many situations because quantities of processes that are naturally considered as algorithmic do not satisfy this restriction. That is why, a problem of extension of ordinary models of algorithm has been very essential for a long period of time. The solution has been given by elaboration of the super-recursive algorithms that allow obtaining results even in the cases when the operation of the device realizing the algorithm is not terminated. It makes possible to increase to a great extent performance abilities of systems of algorithms.

The first models of super-recursive algorithms were suggested and studied in the simultaneously published papers of E.M.Gold (1965) and H.Putnam (1965). These models had the form of limit recursive and partial limit recursive functions. They introduced the concept of computability in a limit (or, as it is more correctly to call it, *inductive computability*) and were aimed at a description of the computations in the limit as well as other algorithmic processes (e.g., learning a language or development of mathematics). At the first glance, they did not look like real algorithms because no computing device was corresponded to the functional representation of algorithms introduced by Gold and Putnam. An existence of such a correspondence is usually included (explicitly or implicitly) in the descriptive definitions of algorithms. Consequently, description of some, at least abstract, device, realizing algorithm, is considered as a necessary condition for a definition of algorithm.

Nine years later another formal construction was suggested for an algorithmic representation of the limit computations (Freyvald 1974). It was a Turing machine with two tapes for which the result of its functioning was defined in a similar way to that in (Gold 1965; Putnam 1965). It was functionally (but not procedurally) equivalent to the functional models from (Gold 1965; Putnam 1965).

Definition 1 (Burgin 1980)**.** *Two algorithms* A *and* U *are called functionally equivalent if they define the same function.*

However, other mathematical constructions (e.g., inductive inference, learning in the limit and so on) that described various infinite algorithmic processes have been elaborated. Some of them have been extensively studied. For example, now inductive inference is a flourishing field in the theory of algorithms.

Recently the paper (Hintikka and Mutanen 1998) appeared in which the concepts of the trial-and-error (tae) computability and satisfiability are introduced. This model is based on the standard construction of a Turing machine with two tapes, a working tape and a bookkeeping (result recording) tape. The ordinary (recursive) computability is extended by a more general definition of a result of computation. That is, the described above Turing machine **T** computes the value of a function *f* for an argument *x* if: 1) **T** begins its process of computation with *x* written on the working tape; 2) the content of the result recording tape will not change from some finite stage of the computation. This content *a* of the result recording tape is called the result of applying **T** to *x,* or the value *f(x)*, i.e., in this case *f(x)=a.* A thorough methodological analysis of the concept of tae-computability and some theoretical results are given in (Hintikka and Mutanen 1998). It is emphasized that the extra condition of finiteness of computational processes looks unnatural if it is considered as knowledge of how far the search for the value *f(x)* need to be carried out.

Being more powerful than ordinary (or recursive) algorithms, such new models of computability are called super-recursive algorithms. Limit Turing machines, introduced in (Burgin 1983; 1992), constitute the most powerful class of the super-recursive algorithms known now. Here we consider a special class of limit Turing machines. They are called, inductive Turing machines (Burgin 1983). It is demonstrated that inductive Turing machines with recursive memory are such super-recursive systems that are the closest to the ordinary (or recursive) algorithms (Burgin and Borodyanskii 1991).

3. INDUCTIVE TURING MACHINES

An inductive Turing machine **M** is a triad (**H**, **Q**, **K**) where **H** is the object domain of **M ,** **Q** is the state domain of **M,** and **K** is the memory domain, or the structured memory of **M** . All these domains are structured. The first of these domains has two structures: the generating and operating ones. If the set H of elements of **H** is some formal language the set of words of which is denoted by L, then its generating structure has the form (A, G, L) where A is the alphabet of L and G is the set of rules generating words from L using the symbols from A. In a general case, the generating structure of **H** determines how the objects from **H** are generated. The operating structure of **H** has the form (H, R, H) where H is the set of elements of **H** and R is the set of operations by which the machine **M** transforms elements from H. As a rule, inductive Turing machine works with words of some formal language, i.e., in this case H = L while its rules are similar (but not identical) to those of the (ordinary) Turing machines.

The state domain **Q** also has two similar structures: the generating and operating ones. If **Q** is a formal language L, then its generating structure has the form (A,G,L) where A is the alphabet of L and G is the set of rules generating the words from L. In a general case, the generating structure of **Q** determines how the objects from **Q** representing states of the machine **M** are generated. The operating structure of **Q** has the form (Q, J, Q) where Q is the set of elements of **Q** and J is the set of the state transitions of the machine **M** .

The memory domain **K** also has two structures: the connectivity and functional ones. The connectivity structure has the form (K, M, K) where K is the set of elements of **K** and M is a binary relation of a determined type. For example M may be a recursive relation. The functional structure defines a partition of K related to the functions of **K**. For example, **K** usually has three parts: input, output, and working memories.

In contrast to an ordinary Turing machine, an inductive Turing machine does not need to stop to produce a result of a computation. Such a result is obtained when after some step of the computation the state of the output memory does not change independently of what is going on in other parts of the whole memory. The computing abilities of the inductive Turing machines are demonstrated with respect to such object domains as sets of words or numbers defined by formulas. Such sets are, as a rule, treated in the theory of algorithms and computations as universal object domains for algorithms.

Let R be an n-ary relation on **H**, that is, R is a subset of the direct product H^n.

Definition 2. *R is called recursive (inductive) if there is some (inductive) Turing machine* **T** *such that* $\mathbf{T}(r) = 1$ *if* $r\in R$ *and* $\mathbf{T}(r) = 0$ *if* $r\notin R$.

Definition 3. *R is called recursively (inductively) computable) if it is computable by some recursive algorithm, i.e., some (inductive) Turing machine* **T** *computes all elements from R and only such elements.*

Remark 1. The tae-computability (Hintikka and Mutanen 1998) as well as the model elaborated in (Freyvald 1974) are special cases of inductive computability.

Lemma 1. *Any inductive relation is inductively computable.*

Remark 2. For recursive relations a similar result is proved in the standard theory of algorithms.

Definition 4. *A quantifier Q is called (inductively) recursively computable if for any recursive relation* R *on* **H** *with* n *free variables including* x *the relation* QxR *is computable by some (inductive) Turing machine.*

Remark 3. Here we consider not only ordinary quantifiers $\exists$, $\forall$ of the classical mathematical logic but arbitrary quantifiers in the sense of (Burgin 1986).

Proposition 1. *The following quantifiers are inductively computable:* $\forall$ *- for all*, $\forall\forall$ *- for almost all, i.e., for all but the finite number;* $\forall_n$ *– for all but n*; $\exists$ *- it exists*; $\exists_n$ *– there exist n objects such that*; $\exists_{>n}$ *- there exist more than n objects such that*; $\exists_{<n}$ *- there exist less than n objects such that.*

Remark 4. In contrast to this result, the quantifiers $\exists$, $\exists_n$, $\exists_{<n}$ are recursively computable while the quantifiers $\forall$, $\forall_n$, $\forall\forall$, $\exists_{>n}$ are recursively non-computable.

4. EFFICIENCY OF SUPER-RECURSIVE ALGORITHMS

Proposition 1 demonstrates that the inductive computability is more powerful than the recursive computability. Really, by Proposition 1, there are such relations R in H^n that the set defined by the expression $\forall xR$ is computable by an inductive Turing machine and non-computable by any (conventional) Turing machine. But it is only a part of the diverse possibilities of super-recursive algorithms. In other words, inductive Turing machines can solve problems unsolvable by recursive algorithms. However, the next result makes possible to demonstrate that the power of the inductive Turing machines is much bigger than that of the recursive algorithms as well as of those super-recursive algorithms that have been introduced prior to the construction of the inductive Turing machines.

Let R be a recursive (recursively computable) n-ary relation on **H**, $m\leq n$, and Q_1 , … ,Q_m be inductively computable quantifiers.

Theorem 1. *The relation* $Q_1x_1 \dots Q_mx_mR$ *is inductively solvable (computable) for any m.*

Remark 5. The theorem, that states that all sets in the arithmetical hierarchy are inductive (thus, being inductively computable) is proved in (Burgin 1983). This theorem is a direct corollary of Theorem 1..

Remark 6. This result makes possible to achieve a new understanding of the famous Gödel incompleteness theorems. According to Theorem 1,formulated above, the statement of the first Gödel incompleteness theorem is not an ultimate law that states existence of such true statements in a system containing formal arithmetic that cannot be proved in this system. The real meaning is that in the historical process of mathematical practice different methods of truth definition for mathematical propositions have been developed. One of them is the usual deduction or proof in formal systems. This method is connected with a universal class **U** of recursive algorithms that are working with mathematical expressions.

Another method of reasoning is related to some class **A** of inductive Turing machines. In this context the first Gödel incompleteness theorem means that **U** is a proper subclass of **A**. Consequently, recursive algorithms are insufficient for truth adjustment of developed mathematical systems. Only super-recursive algorithms can solve this problem. The same is true for the second Gödel incompleteness theorem demonstrating impossibility of a consistency proof inside a theory for developed mathematical systems (such as arithmetic or set theory).

In addition to the higher computational power, the class of inductive Turing machines contains more efficient algorithms than any class of recursive algorithms. It is demonstrated by the following result.

Let **A** be some class of recursive algorithms working with words in some alphabet.

Theorem 2. *For any algorithms A from* **A**, *there is an inductive Turing machine* **M** *that is functionally equivalent to A and has the linear time complexity.*

Time complexity reflects the speed of computations. Consequently, Theorem 2 states that for any recursive algorithm *A,* it is possible to find such an inductive Turing machine **M** that computes the same functions as *A* but with much greater speed if the function is sufficiently complex.

5. CONCLUDING REMARKS

The concept of the inductive computability suggested in (Gold 1965; Putnam 1965) as well as its development in (Burgin 1983) are rooted in the constructions of non-standard analysis (Robinson 1966) and inductive definition of sets (Spector 1959). At the same time, inductive computability is a partial case of the topological computability defined in (Burgin 1983) being the case when the output domain has the discrete topology.

However, it is a mistake to think that super-recursive algorithms are always purely theoretical constructions. There are many processes which are algorithmic but for which super-recursive algorithms provide much more adequate models than recursive algorithms. As an example we can take computations with real numbers. Methods used in numerical analysis are super-recursive algorithms that are only approximated by recursive algorithms. Such constructions have been used in the definition of constructive real numbers (Rice 1951; Mostowski 1957). Numerical methods form a class of super-recursive algorithms distinct from inductive Turing machines because they are working in a domain with a continuous topology. These algorithms are limit Turing machines that work with a domain having the topology of the field of real numbers.

A biological population gives another example, where super-recursive algorithms are important as a tool of investigation. Simulation of their functioning essentially involves infinite processes though contemporary methods of modeling in biology and ecology ignore this fact. Consequently, utilization of super-recursive algorithms provides new powerful facilities for simulation of such processes.

The same is true for investigation, evaluation, and simulation of social processes (Burgin 1993) or for social, political, and/or ecological monitoring.

Many optimization problems, which are solved with or without an aid of a computer, also demand super-recursive representation (Burgin and Shmidskii 1996).

Super-recursive algorithms better than recursive algorithms describe even functioning of a modern computer. Really, when a programmer or user is working with the monitor, the result exists on the screen only when the computer is operating. If the computer stops, then the result on its screen disappears. It means that a result exists only when the algorithm of computer functioning continues its work. This contrasts the condition on ordinary (recursive) algorithms that demands to stop to give a result.

However, working with a printer, we reduce super-recursive algorithms to recursive algorithms. It demonstrates that recursive algorithms are sufficient for modeling those computers and programs that produce their final results by means of a printer. This situation was true for all computers many years ago but it does not correspond to reality now. However, people, considering only intermediate results, reduce super-recursive algorithms to the recursive ones.

In spite of all benefits of super-recursive algorithms, any modern computer (even the fastest one) cannot cram an infinite number of computations in bounded time. It is a challenge for computer engineers to create such a computer that will be able to do this (Stewart 1991). However, it appears to be theoretically possible within classical mechanics. As the matter of fact, it has been proved (Xia 1988*)* that systems of point masses under Newtonian gravitation law can hurl themselves off to infinity in finite period of time.

References

Burgin, M.S. 1980. "Functional equivalence of operators and parallel computations." *Programming*, N6: 3-16 (translated from Russian)

Burgin, M.S. 1983. "Inductive Turing machines." *Notices of the Academy of Sciences of the USSR*, 270, N6: 1289-1293 (translated from Russian)

Burgin, M.S. 1983a. "Limit Turing machines." *Abstracts of the AMS*, v. 5, N3: 309

Burgin, M.S. 1986. "Quantifiers in the Theory of Properties." In *Non-standard Semantics of Non-classical Logics*, Moscow, 99-107 (in Russian)

Burgin, M.S. 1987. "The notion of algorithm and the Turing-Church's thesi*s"*, In *Proceedings of the VIII International Congress on Logic, Methodology and Philosophy of Science*, Moscow, v. 5, part 1, 138-140

Burgin, M.S. 1988. "Arithmetic hierarchy and inductive Turing machines." *Notices of the Academy of Sciences of the USSR*, 299, N3: 390-393 (translated from Russian)

Burgin, M.S.,and Yu.M. Borodyanskii. 1991. "Infinite Processes and Super-recursive Algorithms*." Notices of the Academy of Sciences of the USSR*, 321, N5: 800-803 (translated from Russian)

Burgin, M.S. 1992. "Universal limit Turing machines." *Notices of the Russian Academy of Sciences*, v.325, N4: 654-658 (translated from Russian)

Burgin, M. 1993. "Procedures of sociological measurements." In *Catastrophe, Chaos, and Self-Organization in Social Systems*, Koblenz, 125-129

Burgin, M. and Ya. Shmidskii. 1996. "Is it Possible to Compute Non-computable or why Programmers Need the Theory of Algorithms." *Computers & Software*, N5: 4-8 (in Russian, in collaboration)

Freyvald, R.V. 1974. "Functions and functionals computable in limit." In *Transactions of Latvijas Vlasts Univ. Zinatn. Raksti*, v.210, 6-19 (in Russian)

Gold, E.M. 1965. "Limiting Recursion." *Journal of Symbolic Logic*, 30, N1: 28-46

Hintikka, Ja. and A. Mutanen. 1998. "An Alternative Concept of Computability." In *Language, Truth, and Logic in Mathematics,* Dordrecht: 174-188

Mostowski, A. 1957. "On Computable Sequences." *Fundamenta Mathematicae,* XLIV : 12-36

Putnam, H. 1965. "Trial and Error Predicates and the Solution to a Problem of Mostowski*." Journal of Symbolic Logic*, 30, N1: 49-57

Rice, H.G. 1951. "Recursive Real Numbers." Proceedings of the AMS, 5: 784-791

Robinson, A. 1966. *Non-standard Analysis*, Amsterdam, North Holland

Spector C. 1959. "Inductively Defined Sets of Natural Numbers." In *Infinitistic Methods*, New York, Pergamon Press, 97-102

Stewart, I. 1991. *The Dynamic of Impossible Devices*, Nonlinear Science Today, v.1, N4: 8-9

Xia, Z. 1988. *On the existence of non-collision singularities in Newtonian n body systems*, Doctoral Dissertation, Northwestern University