Theory and practice coexist in the computing world, however, theory has surpassed practice and practice can learn a great deal from the trip.

❖ MARK BURGIN

How We Know WHAT TECHNOLOGY CAN DO

"The biggest advances will come not from doing more and bigger and faster of what we are already doing, but from finding new metaphors, new starting points."

Terry Winograd
Beyond Calculation: The Next 50 Years of Computing, 1997

THE NEW METAPHOR, AS PREDICTED BY WINOGRAD four years ago, does indeed exist. In fact, we only need to learn how to use it in order to open the world of computers to possibilities once unimaginable. The objective of this article is to light the way to mastering that skill, first by illustrating how mathematics has explicated and evaluated computational possibilities by sketching exact boundaries for the world of computers. It is a very complex and sophisticated world. It involves many interacting issues: social and individual, biological and psychological, technical and organizational, economical and political. However, humankind in its development created a system of intellectual devices for dealing with overcomplicated diversities. This system is called "science" and its devices are "theories."

When people want to view what they cannot see with their own eyes, they use magnifying devices. To visualize what is situated far from them, people use telescopes. To discern very small things, such as microbes or cells of living organisms, people use microscopes. In a similar way, theories are magnifying devices for the mind. They may be utilized both as microscopes and telescopes. Because these theoretical devices are very complex, they must be used by experts.

All developed theories have a mathematical core. So, it is not surprising that mathematics has its theories for computers and computations. The main one is *theory of algorithms and computation*. It explains in a logical way how computers function and how computations are organized. It provides a means for evaluating and developing computers, networks, and all computational processes.

Thus, a search for new kinds of computing resulted in DNA and quantum computing. At this point, however, both these paradigms appear to be restricted to specialized domains (DNA for large combinatorial searches, quantum for cryptography) and there are no working prototypes of either. Theory of algorithms finds a correct place for them in a wide range of different computational schemes. Therefore, the first aim of this article is to show how theory has evaluated computational possibilities, sketching exact boundaries for the world of computers.

The second objective of this article is to explain how mathematics has extended these boundaries for computation immensely. What is impossible from the point of view of traditional mathematics—and has been argued as absolutely unworkable—becomes attainable in light of new mathematics. A new mathematical direction that opens powerful and unexpected possibilities is *theory of super-recursive algorithms and computation.*

The third aspiration is to show how to utilize new possibilities. To achieve this, we introduce a new paradigm for computation. It gives better insight into the functioning of computers, networks, and the mind; opening new perspectives for artificial intelligence. Moreover, this form of computation will eclipse the more familiar kinds and will be commercially available before exotic technologies such as DNA and quantum computing.

Some people claim that practice leaves theory

ROBERT NEUBECKER

behind and theory has only to explain what practice has already gained. It is not so with theory of algorithms. Now chemists are designing only the simplest computational units on the molecular level, while theory of parallel computations, comprising molecular computing, has many advanced results. Physicists are only approaching quantum computers, while theory of quantum computations has many results, demonstrating its efficiency over contemporary computers. Today, practice must catch up with theory and it is urgent to know how to bridge the existing gap. This gives a positive answer to Dijkstra's question: "The End of Computing Science?" [5]. On the contrary, science is more alive than ever before. The problem is most professionals don't know it.

Computing through the Microscope of Theory of Algorithms

Mathematics is a powerful tool that helps people achieve new goals as well as understand what is impossible to do. When our experience fails, we have to go to a good theory or elaborate a new one. This theory, like a microscope, allows us to see what is indistinguishable by common sense. That is why it became a task of mathematics to help build more powerful computers and to find the boundaries in computing power. This is the task of computer science with theory of algorithms as its nucleus.

Without an adequate theory, common sense often leads people to fallacies. Even empirical knowledge, which is more advanced than common sense, may be misleading for the best experts when they ignore theory. For example, let us consider the debugging problem. Everybody knows we need precise programs to achieve reliable computer functions. Computer crashes can result in anything from mild inconveniences to the loss of human lives. However, it is practically impossible to write a mistake-free program. And because vital debugging problems are so complicated, it is only natural to search for ways in which computers can debug programs. Programmers worked on this challenge for many years. They suggested different theoretical and empirical methods or used heuristic procedures, but were not able to solve the problem of computerized debugging. Can a theory of algorithms help programmers in their quest?

To our regret, theory of algorithms states "it is impossible to write a program that can debug any other program." This is a consequence of theoretical fact that the halting problem for Turing machines (TMs) is insolvable.

To remedy this, theoreticians suggested using logic for proving program correctness because logic is a powerful tool. However, theory of algorithms enlightens us that it is impossible to prove program correctness for all programs.

Some may get an impression that theory of algorithms provides only negative results. Not true. One of the brightest examples is its contribution to computer architecture. The history of computers tells us that when the first electronic computer was created in the U.S., those responsible invited the great mathematician John von Neumann to review it. Being introduced to the principles of its functioning, he suggested a more advanced computer architecture, now called the "von Neumann architecture." For a long time, all computers had the von Neumann architecture despite the fact that all the other computer components (hardware, software, interfaces) changed very rapidly. However, few know this architecture copied the structure of a TM. von Neumann himself did not explain this, but being an expert in theory of algorithms, he knew TMs intimately.

> HUMAN CREATIVITY multiplied by the computing power of super-recursive devices and algorithms will cause the real revolution in information technology and in our lives.

Today, the theory of algorithms helps solve such vital practical problems as Web reliability, communication security, computer efficiency, and many others. The algorithm has become one of the central concepts of mathematics and a general scientific and technological notion.

Algorithms are something we use every day, sometimes even without much conscious thought. When we want to know time, we look at a watch or clock. This simple rule is an algorithm. All computers function according to algorithms because their programs are algorithms written in programming languages.

An algorithm is a specific kind of recipe, method, or technique for doing something. Informally, an

algorithm is text giving unambiguous (definite) and easy to follow (effective) prescriptions (instructions) how from given inputs (initial conditions) necessary results are derived.

An informal notion of algorithms is comparatively vague, flexible, and simple to treat. Consequently, it is insufficient for an exact study. In contrast, mathematical models are precise, rigid, and bounded. Consequently, they capture only some features of informal notions. To get a better representation, we need to develop mathematical models constantly.

An important peculiarity of the exact concept of an algorithm is it exists in various forms: TMs (deterministic with one tape, with several tapes, with several heads, with n-dimensional tapes; non-deterministic, probabilistic, alternating, and so on), partial recursive functions, finite automata, vector and array machines, neural networks, RAM, among others. Some of these models (such as recursive functions) are only rules for computing. Others (such as TMs or neural networks) give descriptions of a computing device.

The most popular model, suggested by Alan Turing, is called a "deterministic Turing machine" (dTM). Its parts include the *control unit,* which contains rules for functioning and has different states that are changed during computation; the *operating unit,* or the head of a TM; and *memory,* which forms a potentially infinite tape (or several tapes) divided into cells.

We compare classes of algorithms by their computing power. A weaker class of algorithms has less computing power because it allows for computing fewer functions. Two classes of algorithms are equivalent if they can compute the same class of functions. For example, the class of all finite automata is weaker than the class of all dTMs. It means a dTM can compute everything that can compute finite automata, but finite automata cannot compute some functions computable by a TM.

Theory of algorithms makes it possible to evaluate the computing power of new computing schemes. There are several potential approaches to increase the power of computers that go in chemical, physical, and mathematical directions. The first is molecular computing, the most popular branch of which is DNA computing [5]. Its main idea is to design such molecules that solve computing problems. The second direction is even more popular than the first. The main idea behind quantum computing [7] is to perform computation on the level of atoms. The third direction is theory of super-recursive algorithms [3], which is based on a new paradigm for computation that changes computational procedure.

However, molecular and quantum computing can do no more than TMs can do theoretically. For example, quantum computers are only some kinds of nondeterministic TMs, while TMs with many tapes and heads model DNA and other molecular computers. DNA and quantum computers will eventually be only more efficient. In practical computations, they can solve more real-world problems than TMs. However, any modern computer can also solve more real-world problems than a TM because these abstract devices are very inefficient.

All models of algorithms equivalent to the class of all dTMs are recursive algorithms. Stronger models are super-recursive algorithms.

For many years all attempts to find models of algorithms stronger than TMs were fruitless. It influenced emergence of the famous Church–Turing Thesis (CTT): "Any problem that can be solved by an algorithm can be solved by some Turing machine and any algorithmic computation can be done by some Turing machine."

CTT is utilized extensively in theory of algorithms as well as in the methodological context of computer science. It has become almost an axiom. However, this is only one side of the computer coin. Dennis Tsichritzis explored the other side in [11]: "At the beginning of the last century there was a lot of discussions of what computing means. Church closed that discussion with his thesis in 1936. From then on computing was what TM or other similar models can compute. This type of computing was always limited. Most of the interesting problems were proved to be in general insolvable. Nevertheless, they were practically solved using different kinds of heuristics. We come again at a crossroad... I claim that other notions of computing should not only be investigated conceptually but they should be implemented in practice."

New Perspectives through the Telescope of Theory of Algorithms

Theory can help explain what is going on and, like a telescope, help us see far ahead. However, theory has these abilities only when its achievements go ahead of practice. In fact, theory of super-recursive algorithms has this potency.

The first super-recursive algorithms—that is, limit recursive functions—were introduced in 1965 by Gold and Putnam, whose ideas gave birth to the direction called "inductive inference," which is very fruitful in machine learning.

In 1983, independent of inductive inference and

limit recursion, I introduced more powerful inductive Turing machines (ITMs) as algorithms that are not only recursive but super-recursive. Their implications for CTT and the famous Gödel incompleteness theorem were considered refuting CTT and changing the understanding of the theorem.

To comprehend the situation, let us look at conventional models of algorithm. These models include an extra condition where the algorithm stops to give a result. It looks natural because you have to do more after you have what you want. However, we have to change our mind when analyzing real computers.

No computer works without an operating system (OS). Every OS is a program, that is, an algorithm according to the general understanding. But while a recursive algorithm has to stop to give a result, we cannot say that a result of an OS function is obtained when the computer stops functioning. On the contrary, when the computer is out of service, its OS does not give an expected result. Although, from time to time an OS sends some messages (strings of words) to a user, the real result of an OS is the reliable functioning of the computer. Stopping the computer is only a partial result. The real result of an OS is obtained when the computer does not stop (at least, potentially). Thus, we conclude it is not necessary for an algorithm to halt to produce a result.

So far, so good, but how do we determine a result when the algorithm does not stop functioning?

Mathematicians found an answer to this question. Indeed, there are different ways to define a result. Here is the simplest case:

The simplest realistic inductive Turing machine (sITM) has the same structure as a TM with three tapes: input, working, and output. Both inductive and ordinary, a TM performs similar steps of computation. The difference is in output. A TM produces a result only when it halts. The result is a word on the output tape. A sITM produces its results without stopping. It is possible that in the sequence of computations after some step, the word on the output tape is not changing, while sITM continues working. This word, which is not changing, is the result. Thus, sITM does not halt, producing a result after a finite number of computing operations.

Working without halting, ITM can occasionally change its output as it computes more. However, a machine that occasionally changes outputs does not deter humans. They can be satisfied when the printed result is good enough, even if another (possibly better) result may come in the future.

To show that ITMs are more powerful, we need to find a problem solvable by an ITM and insolvable by a TM. This is the halting problem for an arbitrary TM. Turing proved that no TM can solve this problem for all TMs. However, even a sITM can do this [1]. The good news is ITMs give results in a finite time, and are more powerful and efficient [1, 2] than TMs and other recursive algorithms. The bad news is ITMs do not inform when they get their results. This property makes them far better models for many real-life systems and processes.

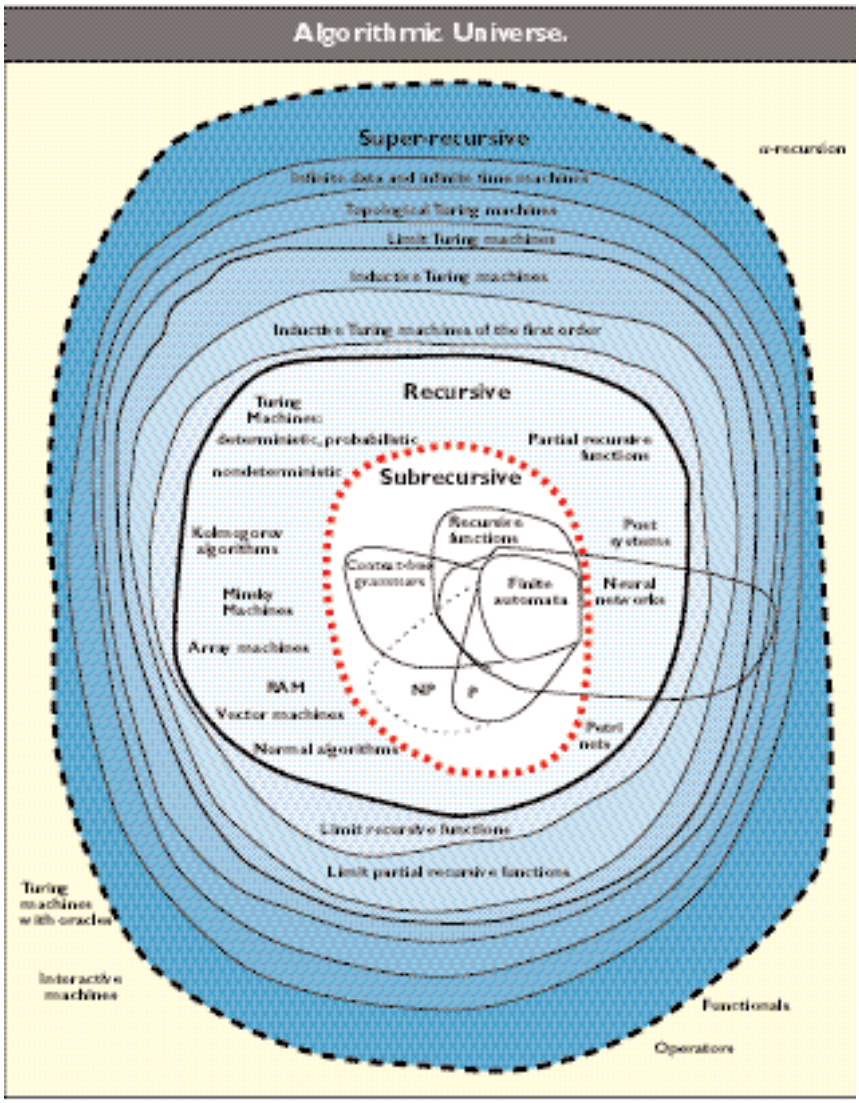
The development of structure allows ITMs to achieve much higher computing power than sITMs. This contrasts TMs: We cannot get greater computing power by changing their power.

From Theory to Practice

We may wonder if the super-recursive approach will be achieved in some distant (if any) future or do we have an existing practice we simply do not understand. To our surprise, we find that people do not completely understand how computers function. An analysis demonstrates that while recursive algorithms gave a correct theoretical representation for computers at the beginning of the computer era, super-recursive algorithms are more adequate for modern computers.

At the beginning, it was necessary to print out data to get a result. After printing, the computer stopped functioning or began to solve another problem. Now people are working with displays. A computer produces its results on the screen. Those results exist on the screen only if the computer functions. This is exactly the case of ITMs because its results are obtained without stopping. The option to print some results and switch off the computer only shows that recursively algorithms can be modeled by ITMs.

However, misunderstanding the way computers work is not unique. For thousands of years, people thought the Sun rotated around the Earth. It was



not until the 16th century that Copernicus proved differently.

Huge networks like the Internet give another important example when conventional algorithms are not adequate. Algorithms embodied in a multiplicity of different programs organize network functioning. A recursive algorithm must stop to give a result, but if a network shuts down, we get no results. Of course, this means something needs fixing.

The conventional paradigm is based on our image of computer utilization, which consists of three stages: formalizing a problem, writing a computer program, and obtaining a solution by program execution. After this, you either stop your computer or begin to solve another problem.

This process is similar to the usage of a car. We go some place by car, then possibly go to another place, then another, and so on. At some point, however, we park the car and for a definite time do not use it. This is called the "car paradigm" because some object is utilized only periodically for achieving some goal. After the goal is achieved, the object does not function (at least for some time).

Now, consider how people use clocks. After buying a clock, a person sets it up and turns it on initially and then the clock functions until it breaks. People look at the clock from time to time (no pun intended) to determine the hour. We call this the "clock paradigm" when some object is functioning all the time without stopping, while those who utilize it only do so periodically to get some results from it. Recursive algorithms imply that modern computers are utilized according to the car paradigm, while super-recursive algorithms suggest the clock paradigm.

The development of information technology gave birth to systems that incorporate many computers and other electronic devices, for example, the Internet. These systems possess new properties. Today, we could not imagine the Internet would stop functioning for even a short period of time. Thus, the Internet works according to the clock paradigm. Consequently, only super-recursive algorithms such as ITMs—ordinary and interactive—can model such systems correctly.

Although some computers and networks function in the clock paradigm, conscious application of the new approach provides several important benefits. First, it gives a better understanding of computational results obtained during some finite period of time. Second, it shows how to utilize computers in a better way. Third, it makes it possible to use more adequate theoretical models for investigating and developing computers. For example, simulation and/or control of many technological processes in industry is better modeled when these processes are treated as potentially infinite.

People must be creative in order to utilize higher possibilities of the new paradigm. Creativity will continue to be a clue to the highest achievements, but super-recursive computations will increase these heights to an unimaginable degree. Humans and computers must be cooperating systems. Human creativity multiplied by the computing power of

WITHOUT AN ADEQUATE THEORY, COMMON SENSE often leads people to fallacies. Even empirical knowledge, which is more advanced than common sense, may be misleading for the best experts when they ignore theory.

super-recursive devices and algorithms will cause the real revolution in information technology and in our lives.

The current situation is reflected in this metaphor: Primitive planes exist, but most people do not know how to use them for flying, so they use them as cars. A new theory explains how to fly these planes and how to build much more advanced planes. This implies people will need new skills because flying is different from driving.

Conclusion

In business and industry, the main criterion is enterprise efficiency that, in economics, is called "productivity." Consequently, computers are important not because they make computations with more speed than before. What really matters is computers can increase productivity. Reaching higher productivity depends on improved procedures. Without proper procedures and necessary skills, technical devices can decrease productivity. Consequently, methods that develop computer procedures are more important than improving hardware or software. In other words, DNA computing is like a new car model. Quantum computation is like the first airplane. Super-recursive computation is like a rocket that can take people beyond the Church–Turing Earth.

Such rockets might take us to the Moon and beyond if we learn how to navigate them. However, we will need new physical ideas for realization of super-recursive algorithms to a full extent. Using our metaphor, we may say that spaceships able to take us to the stars are still in their infancy.

There are several models of super-recursive algorithms. The main advantage of ITM is it works with finite objects and obtains the results in a finite period of time. Other models either work with infinite objects (for example, TMs, neural networks, and topological algorithms working with real numbers [2, 8, 9]) or need infinite time to produce results beyond CTT (for example, infinite time TMs [7] and persistent TMs [6]).

If we imagine an Algorithmic Universe (see accompanying figure), we see its recursive region is a closed system that entangles depressing incompleteness results such as Gödel incompleteness theorems. In contrast, the super-recursive region of the Algorithmic Universe is open, implying development, creativity, and limitless human endeavor.

To conclude, I should note that theory of super-recursive computing has an important message for society as well as technology. It only looks smart to stop and to enjoy after you get something. Theory of super-recursive computing says it is better to continue to be active; certainly an admirable theory for life. C

References

1. Burgin, M. Super-recursive algorithms as a tool for high performance computing. In *Proceedings of the High Performance Computing Symposium*. (San Diego, Calif., 1999), 224–228.
2. Burgin, M. Topological algorithms. In *Proceedings of 16th International Conference for Computers and Applications ISCA.* (Seattle, WA, 2001), 61–64.
3. Cho, A. Hairpins trigger an automatic solution. *Science* 288 (2000). AAAS, Washington, D.C.
4. Deutsch, D., Ekert, A., and Lupacchini, R. Machines, logic and quantum physics. *Bulletin of Symbolic Logic 6* (2000). Assoc. of Symbolic Logic, Poughkeepsie, NY, 265–283.
5. Dijkstra, E.W. The end of computing science? *Commun. ACM 44,* 3 (Mar. 2001), 92.
6. Goldin, D. and Wegner, P. Persistent Turing machines. Brown University Technical Report, 1998
7. Hamkins, J.D., and Lewis, A. Infinite time Turing machines. *J. Symbolic Logic 65* (2000). Assoc. of Symbolic Logic, Poughkeepsie, NY, 567–604.
8. Moore, C. Recursion theory on the real- and continuous-time computation. *Theoretical Computer Science* 162 (1996), 23–44
9. Siegelman, H.T. *Neural Networks and Analog Computation*. Birkhauser, Berlin, 1999
10. Tsichritzis, D. Forget the past to win the future. *Commun. ACM 44,* 3 (Mar. 2001), 100–101.
11. Winograd, T. The design of interaction. *Beyond Calculation: The Next 50 Years of Computing* (1997) Copernicus, N.Y.,149–161

Mark Burgin (mburgin@math.ucla.edu) is a visiting scholar in the Department of Mathematics at the University of California, Los Angeles, CA.