1

A CRITICAL REVIEW OF THE NOTION OF THE ALGORITHM IN COMPUTER SCIENCE

Computer science inherited its present conceptual foundations from a branch of pure mathematics that, historically, had been exploring the fundamental nature of mathematical computation since before the turn of the century. It is argued that the conceptual concerns of computer science are different from the conceptual concerns of mathematics, and that this mathematical legacy, in particular the notion of the algorithm, has been largely ineffective as a paradigm for computer science. It is first necessary to understand the role of the algorithm in mathematics.

1.1 THE NOTION OF THE ALGORITHM IN MATHEMATICS

The notion of the algorithm is fundamental to mathematics. To understand the significance of the algorithm to mathematics, it is necessary to understand the history of its development. The term derives from the name of an important ninth-century Persian mathematician, Mohammed ibn Musa al-Khowarizmi, who in about AD 825 wrote a small book describing how to calculate with a new ten-symbol, positional value number system developed in India. It described simple procedures for carrying out addition, subtraction,

© ACM, 1993. This chapter is a minor revision of my work: K. Fant, "A Critical review of the notion of the algorithm in computer science," *Proceedings of the 21st Annual Computer Science Conference*, February 1993, pp. 1–6.

multiplication, and division in the new system. Around 1120 this small book was translated into Latin under the title *Liber Algorismi de numero Indorum* (The Book of al-Khowarizmi on the Hindu number system). This translation was widely distributed and introduced the Hindu-Arabic number system to Europe. By the mid-thirteenth century al-Khowarizmi was largely forgotten, and the term algorism (Latin for al-Kowarizmi's book) came generally to refer to computation in the new number system. At this time an algorism was any book related to the subject. The algorisms were the four arithmetic operations. An algorist was one who calculated in the new number system as opposed to an abacist who used an abacus. By 1500 the algorists had prevailed and the abacists had largely disappeared from Europe.

These algorisms were strictly mechanical procedures to manipulate symbols. They could be carried out by an ignorant person mechanically following simple rules, with no understanding of the theory of operation, requiring no cleverness and resulting in a correct answer. The same procedures are taught to grade school children today. Computing with Roman numerals, on the other hand, required considerable skill and ingenuity. There also existed at this time other examples of mechanical formulation such as Euclid's method to find the greatest common denominator of two numbers. The fact that dumb mechanical manipulations could produce significant and subtle computational results fascinated the medieval mathematicians. They wondered if it was possible that the whole of mathematics or even all of human knowledge could be mechanically formulated and calculated with simple rules of symbol manipulation.

Gottfried Leibniz attempted just such a formulation in the 1660s with his calculus ratiocinator or characteristica universalis. The object was to "enable the truths of any science, when formulated in the universal language, to be computed by arithmetical operations" [1]. Arithmetical here refers to the algorisms. Insight, ingenuity, and imagination would no longer be required in mathematics or science. Leibniz did not succeed, and the idea lay fallow for two hundred years.

During this period Euclidian geometry, with its axioms and rules of reasoning from the simple to the complex, continued to reign as the fundamental paradigm of mathematics. In the 1680s, after the invention of analytical geometry, and after he had made new discoveries with his own invention of his fluxional calculus, Sir Issac Newton was careful to cast all the mathematical demonstrations in his presentation of these new discoveries in *Philosophiae naturalis principia mathematica* in classical Euclidian geometry. A symbolic analytical presentation would neither have been understood nor accepted by his contemporaries. Geometry, which deals with relationships among points, lines, and surfaces, was intuitive, obvious, and real. Algebra, which deals with arbitrary symbols related by arbitrary rules, did not relate to any specific reality. While algebra was practical and useful, it was not considered fit territory for fundamental theoretical consideration. Late into the nineteenth-century symbolic computation was distrusted and discounted. This attitude is exemplified by a nineteenth-century astronomer who remarked that he had not the “smallest confidence in any result which is essentially obtained by the use of imaginary symbols” [2].

The dream of formalizing thought in terms of mechanical manipulation of symbols reemerged with the symbolic logic of George Boole presented in his book *Laws of Thought* in 1854. Boole argued persuasively that logic should be a part of mathematics as opposed to its traditional role as a part of philosophy. Gottlob Frege went several steps further and suggested that not only should logic be a part of mathematics but that mathematics should be founded on logic, and he began a program to derive all of mathematics in terms of logic.

Meanwhile the paradigmatic edifice of Euclidian geometry was beginning to show cracks with the discovery of non-Euclidian geometries that were internally consistent and therefore were just as valid mathematical systems as Euclidian geometry. Symbolic computation achieved paradigmatic preeminence with the publication in 1899 of David Hilbert’s characterization of Euclidian geometry in terms of algebra, *Grundlagen der Geometrie* (*Foundations of Geometry*), which emphasized the undefined nature of the axioms. “One must be able to say at all times—instead of points, straight lines and planes—tables, chairs and beer mugs” [3]. Euclidian geometry was after all just one of many possible axiomatic symbolic computation systems.

As the twentieth century dawned, symbolic computation had been established as the arena of mathematical theorizing, and logical axiomatic systems provided the rules of the game. The mathematicians were hot on the trail of settling the game once and for all. They seemed to be on the verge of fulfilling Leibniz’s dream of the universal symbolic language that would proffer absolute certainty and truth. The quest was led by Hilbert who outlined a program to settle once and for all the foundational issues of mathematics. The program focused on the resolution of three questions:

1. Was mathematics complete in the sense that every statement could be proved or disproved?
2. Was mathematics consistent in the sense that no statement could be proved both true and false?
3. Was mathematics decidable in the sense that there existed a definite method to determine the truth or falsity of any mathematical statement? [4]

The definite method of decidability in question 3 was the modern incarnation of Leibniz’s arithmetical operations on his universal symbolic language. Mechanical symbol manipulation reemerges at the very foundations of theoretical mathematics.

Hilbert firmly believed that the answer to all three questions was yes, and the program was simply one of tidying up loose ends. Hilbert was convinced that an unsolvable mathematical problem did not exist, “every mathematical problem must necessarily be susceptible to an exact statement, either in the form of an actual answer to the question asked, or by the proof of the impossibility of its solution" [5].

In 1931 Kurt Godel demonstrated that any axiom system expressive enough to contain arithmetic could not be both complete and consistent in the terms of the axiom system. This result was the death knell for Hilbert's program. The answers to the first two questions were no. There remained the question of decidability, the *Entscheidungsproblem*, as Hilbert named it: the definite method of solving a mathematical problem. After Godel proved that unsolvable problems (unprovable theorems) could exist in an axiom system, the decidability problem became a search for a definite method to determine if a given problem was solvable or unsolvable in a given axiom system.

The decidability problem appealed directly to the notion of a definite method, which was also referred to as an effective procedure or a mechanical procedure. An iterative step-by-step procedure had always been fundamental to mathematics but had been intuitively accepted and had not been a subject of investigation itself. One knows an effective procedure when one sees one. But to demonstrate something about the nature of effective procedures there must be a precise characterization of what constitutes an effective procedure.

Hilbert made it clear what constituted an acceptable mathematical solution in his 1900 paper posing 23 problems that he considered important to the future of mathematics:

> . . . that it shall be possible to establish the correctness of a solution by means of a finite number of steps based upon a finite number of hypotheses which are implied in the statement of the problem and which must always be exactly formulated. [5]

Satisfactorily characterizing this notion of effective or mechanical procedure became an important foundational issue in mathematics and several mathematicians applied themselves to the problem. Among them were Jacques Herbrand and Godel, Emil Post, Alan Turing, Alonzo Church, and A. A. Markov. Each had a different characterization of effective computability, but all were shown later to be logically equivalent. In 1936 both Church with his lambda calculus and Turing with his machine proved that no effective procedure existed to determine the provability or unprovability of a given mathematical problem. The answer to Hilberts third question was also no. Leibniz's calculus ratiocinator with its arithmetical resolution of all questions proved to be not possible. Ingenuity, insight, and imagination could not be done away with in mathematics after all.

Despite the failure of Hilbert's program, questions of effective computability have continued to be a fundamental concern of mathematicians. Through the 1940s and 1950s Markov tried to consolidate all the work of the others on effective computability and introduced the term algorithm with its modern meaning as a name for his own theory of effectively computable functions. In the translated first sentence of his 1954 book *Teoriya Algorifmov* (*Theory of Algorithms*) he states:

> In mathematics, "algorithm" is commonly understood to be an exact prescription, defining a computational process, leading from various initial data to the desired result. [6]

The term algorithm was not, apparently, a commonly used mathematical term in America or Europe before Markov, a Russian, introduced it. None of the other investigators, Herbrand and Godel, Post, Turing, or Church used the term. The term, however, caught on very quickly in the computing community. In 1958 a new programming language was named ALGOL (**ALGO**rithmic **L**anguage). In 1960 a new department of the *Communications of the ACM* was introduced called "Algorithms" [7].

Historically, the algorithm was developed to investigate the foundations of mathematics, and it has evolved in relation to the needs of mathematicians. The notion of the algorithm in mathematics is a limiting definition of what constitutes an acceptable solution to a mathematical problem. It establishes the ground rules of mathematics.

1.2 THE ADVENT OF COMPUTERS

The electronic digital computer emerged in 1945. It computed one step at a time, was by practical necessity limited to a finite number of steps, and was limited to a finite number of exactly formulated hypotheses (instructions). The electronic digital computer was an incarnation of the mathematician's effective solution procedure. The mathematicians, being intimately involved with the creation of the computer, having studied mechanical computation for half a century, and having in hand an explicitly mechanical model of computation in the Turing machine, quite reasonably became the de facto theorists for this new phenomenon. One of these mathematicians, John Von Neumann, was a student of Hilbert's and a significant contributor to his program to resolve the foundations of mathematics. Another was of course Turing himself. The related mathematical concepts along with the notion of the algorithm were transplanted into the fledgling science of computers.

The notion of the algorithm has become accepted as a fundamental paradigm of computer science.

> The notion of the algorithm is basic to all computer programming. . . . [8]
>
> One of the concepts most central to computer science is that of an algorithm. [9]

To appreciate the role of the algorithm in computer science, it is necessary first to characterize computer science.

1.3 COMPUTER SCIENCE

Many attempts have been made to define computer science [10–14]. All these definitions view computer science as a heterogeneous group of disciplines related to the creation, use, and study of computers. A typical definition simply offers a list of included topics: computability, complexity, algorithm theory, automata theory, programming, high-level languages, machine languages, architecture, circuit design, switching theory, system organization, numerical mathematics, artifiial intelligence, other applications, and so forth. The most recent and comprehensive survey of the attempts to define computer science is an article in the *Annals of the History of Computing* [15].

Computer science appears to consist of a quite disparate collection of disciplines, but there is a common thread of conceptual focus running through the various disciplines of computer science. All of the disciplines that are included under the heading of computer science in any list are concerned in one way or another with the creation of or actualization of process expressions. A logic circuit is an expression of a logical process. An architecture is an expression of a continuously acting process to interpret symbolically expressed processes. A program is a symbolic expression of a process. A programming language is an environment within which to create symbolic process expressions. A compiler is an expression of a process that translates between symbolic process expressions in different languages. An operating system is an expression of a process that manages the interpretation of other process expressions. Any application is an expression of the application process.

Computer science can be viewed as primarily concerned with questions about the expression of processes and the actualization of those expressions. What are all the possible ways a process can be expressed? Are some expressions superior in any sense to other expressions? What are all the possible ways of actualizing an expression. Are there common conceptual elements underlying all expressions? What is the best programming language? How can the best program be formulated? How can the best architecture be built? What is the best implementation environment? These are the questions that occupy computer scientists, and they all revolve around the nature of process expression.

Mathematicians, on the other hand, are primarily interested in exploring the behavior of specific processes or classes of process. They bypass general problems of expression by appealing to a very formal and minimalized model of expression, the algorithm as characterized by the Turing machine. They are only interested in whether an expression is possible and whether it conforms to certain specific properties. The mathematicians consider the process as independent of its expression. A process may be expressed in any convenient language and executed on any convenient machine including a human with a pencil.

Mathematics is primarily concerned with the nature of the behavior of process independent of how that process is expressed:

the nature of a process is considered independent of its expression.

Computer science is primarily concerned with the nature of the expression of processes regardless of what particular process might be expressed:

the nature of expression is considered independent of its process.

There is much overlap between the interests of computer science and mathematics, but the core concern with the nature of process expression itself is the unique conceptual focus that distinguishes computer science from the other sciences and from mathematics. Computer science is the science of process expression.

1.4 THE ALGORITHM IN COMPUTER SCIENCE

Introductory texts on computer science often begin with a chapter on the notion of the algorithm declaring it the fundamental paradigm of computer science. Conspicuously absent from these introductory chapters is discussion of how the notion contributes to the resolution of significant problems of computer science. In the remaining chapters of these texts there is typically no further appeal to the notion of the algorithm and rarely even a usage of the word itself. The notion is never or very rarely appealed to in texts on logic design, computer architecture, operating systems, programming, software engineering, programming languages, compilers, data structures, and data base systems.

The notion of the algorithm is typically defined by simply presenting a list of properties that an expression must posses to qualify as an algorithm. The following definition of an algorithm is typical:

1. An algorithm must be a step-by-step sequence of operations.
2. Each operation must be precisely defined.
3. An algorithm must terminate in a finite number of steps.
4. An algorithm must effectively yield a correct solution.
5. An algorithm must be deterministic in that, given the same input, it will always yield the same solution.

This is pretty much what Hilbert proposed in 1900, and it is easy to see how this list of restrictive characteristics serves to define what is acceptable as a mathematical solution. But what conceptual service does the notion of the algorithm perform for computer science?

The notion of the algorithm demarcates all expressions into algorithm and nonalgorithm, but what purpose does it serve to know that one program is an acceptable mathematical solution and another is not? Is the expression of one fundamentally different from the expression of the other? Is one interpreted differently from the other? Are algorithms first-class citizens in some sense and nonalgorithms second-class citizens? Does determining whether or not a given expression is an acceptable mathematical solution aid in building better computer systems or in writing better programs?

Important process expressions do not qualify as algorithms. A logic circuit is not a sequence of operations. An operating system is not supposed to terminate, nor does it yield a singular solution. An operating system cannot be deterministic because it must relate to uncoordinated inputs from the outside world. Any program utilizing random input to carry out its process, such as a Monte Carlo simulation or fuzzy logic simulation, is not an algorithm. No program with a bug can be an algorithm, and it is generally accepted that no significant program can be demonstrated to be bug free. Programs and computers that utilize concurrency where many operations are carried out simultaneously cannot be considered algorithms. What does it mean when a sequential program qualifying as an algorithm is parallelized by a vectorizing compiler, and no longer qualifies as an algorithm.

While a digital computer appears to be an algorithmic machine, It is constructed of nonalgorithmic parts (logic circuits) and a great deal of what it actually does is nonalgorithmic. These difficulties with the notion of the algorithm have not gone unnoticed, and a variety of piecemeal amendments, revisions, and redefinitions have been proposed:

> . . . there is an extension of the notion of algorithm (called nondeterministic algorithm). [11]
>
> Any computer program is at least a semi-algorithm and any program that always halts is an algorithm. [16]
>
> There is another word for algorithm which obeys all of the above properties except termination and that is computational procedure. [17]
>
> An algorithm A is a *probabilistically good algorithm* if the algorithm “almost always” generates either an exact solution or a solution with a value that is “exceedingly close” to the value of the optimal solution. [18]
>
> The procedure becomes an algorithm if the Turing machine always halts. [19]
>
> By admitting probabilistic procedures in algorithms. . . . [20]
>
> . . . if, after executing some step, the control logic shifts to another step of the algorithm as dictated by a random device (for example, coin tossing), we call the algorithm random algorithm. [21]
>
> An algorithm which is based on such convergence tests is called an infinite algorithm. [21]
>
> Algorithms that are not direct are called indirect. [22]
>
> We drop the requirement that the algorithm stop and consider infinite algorithms. [22]

These authors have sensed an inappropriate conceptual discrimination or simply an incompleteness and proposed a remedy. Programs that do not terminate, are not deterministic, and do not give specific solutions can now be "included." They are no longer simply nonalgorithmic, they now have positive labels, but simply assigning labels to nonalgorithms misses the point. The point is that algorithm–nonalgorithm is not a conceptual distinction that contributes to an understanding of process expression.

As a paradigm of process expression, the notion of the algorithm is decidedly deficient. It offers no suggestion as to how an operation might be precisely defined. Nor does it suggest how a sequence should be determined. Data are not even mentioned. The definition simply states that an algorithm must consist of a sequence of precisely defined operations. This unsupported imperative is at once an admission of expressional incompleteness and a refusal to be complete. The other algorithmic properties of termination, correctness, and determination, while important to issues of computation, are quite irrelevant to issues of process expression.

The notion of the algorithm simply does not provide conceptual enlightenment for the questions that most computer scientists are concerned with.

1.5 CONCLUSION

What is essentially a discipline of pure mathematics has come to be called "the theory of computer science," and the notion of the algorithm has been decreed to be a fundamental paradigm of computer science. The mathematical perspective, however, is the wrong point of view. It is asking the wrong questions. Mathematicians and computer scientists are pursuing fundamentally different aims, and the mathematician's tools are not as appropriate as was once supposed to the questions of the computer scientist. The primary questions of computer science are not of computational possibilities but of expressional possibilities. Computer science does not need a theory of computation; it needs a comprehensive theory of process expression.

REFERENCES

1. C. Lejewski. History of logic. In *Encyclopedia Britannica Macropaedia*, Vol. 11. Chicago: William Benton, 1974, pp. 56–72.
2. M. M. Garland. *Cambridge Before Darwin*. Cambridge: Cambridge University, 1980, p. 36.
3. H. G. Forder and F. A. Valentine. Euclidian geometry. In *Encyclopaedia Britannica Macropedia*, Vol. 7. Chicago: William Benton, 1974, pp. 1099–1112.
4. A. Hodges. *Alan Turing the Enigma*. New York: Simon and Schuster, 1983, p. 91.
5. D. Hilbert. Mathematical problems. In *Mathematics People, Problems, Results*, Vol. 1, ed. by D. M. Campbell and J. C. Higgins. Belmont, CA: Wadsworth International, 1984, p. 275.

6. A. A. Markov. *Theory of Algorithms*, trans. by J. J. Schorr-Kon. Jerusalem: Keter Press, 1971, p. 1.
7. J. H. Wegstein. Algorithms. In *Communications of the ACM* 3 (February 1960), p. 73.
8. D. E. Knuth. *Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1969, p. 1.
9. Z. W. Pylyshyn. Theoretical ideas: Algorithms automata and cybernetics. In *Perspectives on the Computer Revolution*, ed. by Zenon W. Pylyshyn. Englewood Cliffs, NJ: Prentice-Hall, 1970. pp. 60–68.
10. S. Amarel. Computer science. In *Encyclopedia of Computer Science* (1st ed. 1976). New York: Petrocelli/Carter, 1976, pp. 314–318.
11. M. S. Carberry, H. M. Khalil, J. F. Leathrum, and L. S. Levy. *Foundations of Computer Science*, Potomac, MD: Computer Science Press, 1979, pp. 2–4, 16.
12. J. M. Brady. *The Theory of Computer Science*. London: Chapman and Hall, 1977, pp. 8–9.
13. A. Ralston. *Introduction to Programming and Computer Science*. New York: McGraw-Hill, 1971, pp. 1–5.
14. I. Pohl and A. Shaw. *The Nature of Computation*. Rockville, MD: Computer Science Press, 1981, pp. 3–7.
15. P. Ceruzzi. Electronics technology and computer science, 1940–1975: A coevolution. *Annals of the History of Computing* 10 (4, 1989): 265–270.
16. R. R. Korfhage. Algorithm. In *Encyclopedia of Computer Science* (1st ed. 1976). New York: Petrocelli/Carter, 1976, p. 49.
17. E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Potomac, MD: Computer Science Press, 1979, pp. 1–2.
18. B. W. Wah and C. V. Ramamoorthy. Theory of algorithms and computation complexity with applications to software design. In *Handbook of Software Engineering*, ed. by C. R. Vick and C. V. Ramamoorthy. New York: Van Nostrand Reinhold, 1984, p. 88.
19. K. Maly and A. R. Hanson. *Fundamentals of the Computing Sciences*. Englewood Cliffs, NJ: Prentice-Hall, 1978, p. 41.
20. F. S. Beckman. *Mathematical Foundations of Programming*. Reading, MA: Addison-Wesley, 1980, p. 398.
21. E. V. Krishnamurthy. *Introductory Theory of Computer Science*. New York: Springer-Verlag, 1983, p. 3.
22. J. K. Rice and J. R. Rice. *Introduction to Computer Science*. New York: Holt, Rinehart and Winston, 1969, pp. 47, 49.