# The Origins of the Turing Thesis Myth

Dina Goldin
Computer Science & Engr. Dept.
University of Connecticut
Storrs, CT, USA
dqg@cse.uconn.edu

Peter Wegner
Computer Science Dept.
Brown University
Providence, RI, USA
pw@cs.brown.edu

June 25, 2004

**Abstract**

In this paper, we identify and analyze the historical reasons for the widespread acceptance of what we call the *Turing Thesis myth*, which extends the Turing Thesis to imply that Turing Machines model all computers. This is the third in a series of related papers by the authors, beginning with [5] and continuing with [6].

## 1 The Turing Thesis myth

Turing's famous 1936 paper [15] developed the *Turing Machine* (TM) model and showed that TMs have the expressiveness of algorithms (Turing Thesis).

**Turing Thesis:** Whenever there is an effective method (algorithm) for obtaining the values of a mathematical function, the function can be computed by a TM.

Turing's thesis has since been reinterpreted to imply that Turing Machines model *all* computation, rather than just functions. This claim, which we call the *Strong Turing Thesis*, can be found in today's popular undergraduate theory textbooks:

**Strong Turing Thesis:** A TM can do (compute) anything that a computer can do.

This bold claim is part of the mainstream theory of computation, yet Turing himself would have denied it. In the same famous paper, he also introduced interactive *choice machines* as another model of computation distinct from TMs and not reducible to it. Choice machines extend TMs to interaction by allowing a human operator to make choices during the computation.

In fact, the Strong Turing Thesis is incorrect – the function-based behavior of algorithms does not capture all forms of computation. For example, as explained in [5], interactive protocols are not algorithmic. Yet the myth of the correctness of the Strong Turing Thesis is dogmatically accepted by most computer scientists. As Denning recently wrote in this publication [7], "we are captured by a historic tradition that sees programs as mathematical functions".

The reasons for the widespread acceptance of the Turing Thesis myth can be traced to the establishment of computer science as a separate discipline in the 1960's. To understand these reasons better, we can identify four distinct claims that make up the myth, and examine them individually:

**Claim 1.** All computable problems are function-based.
Reason: Adoption of mathematical principles for the fundamental notions of computation, identifying computability with the computation of functions, as well as with TMs.

**Claim 2.** TMs can compute any solvable problem.
Reason: Adoption of algorithms as the central and complete concept of computer science, without an explicit agreement on the meaning of this term.

**Claim 3.** TMs can simulate any computer.
Reason: Misattributing general purpose computing to Turing Machines.

**Claim 4.** TMs serve as a general model for computers.
Reason: Misunderstanding the definition of a TM.

In this paper, we investigate these claims and analyze why they have emerged in the computer science community, focusing on computer science textbooks for our primary sources. It is time to recognize that today's computing applications, such as web services, intelligent agents, operating systems, and graphical user interfaces, cannot be modeled by TMs; alternative models are needed. We conclude by considering interaction as an extension of TMs.

## 2 What is Computation?

The first two reasons for the Turing Thesis myth are related to the basic understanding of the notion of computation, as adopted by the Theory of Computation.

### 2.1 The mathematical worldview

The Theory of Computation predates the establishment of computer science as a discipline, having been a part of mathematics before the 1960's. Its founders include such notable mathematicians as Godel, Kleene, Church, and Turing. [1] Mathematicians naturally equated the notion of computability with the computation of functions. Martin Davis's 1958 textbook [4], popular among computer scientists, reflected the *mathematical worldview* that all computation is function-based, and therefore captured by TMs. It begins as follows:

> "This book is an introduction to the theory of computability and non-computability, usually referred to as the theory of recursive functions... the notion of TM has been made central in the development."

In particular, this view assumes that all computation is *closed*. There is no input or output taking place during the computation; any information needed during the computation is provided at the outset as part of the input. These assumptions are embodied by the semantics of TMs.

**Mathematical worldview:** All computable problems are function-based.

The mathematical worldview was enthusiastically adopted by early leaders of the computer science community, including Von Neumann, Knuth, Karp, Rabin, and Scott. Mathematics has been used as a foundation of physics and other scientific disciplines, and it was believed that mathematics could be used as a basis for computer science. Davis's book proved very influential, cementing the acceptance of the mathematical worldview among computer scientists of the 1950's and 1960's.

The mathematical worldview is the first of the four claims that constitute the Turing Thesis myth. It can be contrasted with the *engineering worldview*, where computation is viewed as an

[1] While Turing's training and original contributions were mathematical, we believe that his later work classifies him as a computer scientist rather than a mathematician – perhaps the first one.

ongoing transformation of inputs to outputs – e.g., control systems. This conceptualization of computation allows, for example, the *entanglement* of inputs and outputs; later inputs to the computation may depend on earlier outputs. Such entanglement is impossible in the mathematical worldview, where all inputs precede computation, and all outputs follow it.

### 2.2 Algorithms and computability

An algorithm is a "recipe" for carrying out a computation, one that can be followed mechanically. According to the function-based mathematical worldview adopted by computer science, the computational role of an algorithm is to transform input data to output data:

**Role of algorithm:** Given some finite input $x$, an algorithm describes the steps for effectively transforming it to an output $y$, where $y$ is $f(x)$ for some recursive function $f$.

The notion of an algorithm is inherently informal, since an algorithmic description is not restricted to any single language or formalism. The first high-level programming language developed expressly to help specify algorithms was ALGOL (ALGOrithmic Language). Introduced in the late 50's and refined through the 1960's, it was the standard for the publication of algorithms. True to the function-based conceptualization of algorithms, ALGOL provided no constructs for input and output, considering these operations outside the concern of algorithms. Not surprisingly, this absence hindered the adoption of ALGOL by the industry for commercial applications.

TMs, which transform input strings to output strings, serve as a formal model for algorithms:

**Informal approach:** A problem is *solvable* if it can be specified by an algorithm.

**Formal approach:** A problem is *solvable* if there exists a Turing Machine for computing it.

The coexistence of the informal (algorithm-based) and the formal (TM-based) approaches to defining solvable problems persists to this day. We call it the Turing Thesis corollary, and it is the second of the four claims that make up the Turing Thesis myth:

**Turing Thesis corollary:** TMs can compute any solvable problem.

The legitimacy of this corollary is based on two premises. The first one is the Turing Thesis, which equates function-based computation with TMs. The second one, usually left unstated, is what we call the mathematical worldview – the assumption that all algorithms and computable problems are function-based.

The perceived validity of this corollary was greatly strengthened by the many attempts to find models of computation that are more expressive than TMs, for example extending the number of tapes or reading heads on the machine. All these attempts failed, because they continued to adhere to the mathematical worldview, and never considered problems that are not function-based.

While the Turing Thesis remains true, the mathematical worldview no longer reflects the nature of computational problems. An example of such a problem is *driving home from work* [5]:

**Driving home from work:** create a car that is capable of driving us home from work, where the location of both work and home can be viewed as input parameters.

Assuming that the driving is to take place in a real-world environment, this problem is not computable with a function-based algorithm. Consider the input to such a function. It would have to be detailed enough so the car could predict the direction and strength of the wind at each point in the drive, so as to compensate for it. It should also enable the car to anticipate

the location of all pedestrians, so as to avoid running over them. As we discuss in [16], this is impossible – there is no such computable function. However, the problem *is* computable by a control mechanism, as in a robotic car, that continuously receives video input of the road and actuates the wheel and brakes accordingly.

# 3 What is an Algorithm?

## 3.1 The ACM curriculum

The 1960's saw a proliferation of undergraduate computer science (CS) programs; according to ACM [2], the number of CS programs in the USA increased from 11 in 1964 to 92 in 1969. This increase was accompanied by intense activity towards establishing the legitimacy of this new discipline in the eyes of the academic community. ACM played a central role in this activity. In 1965, it enunciated the justification and description of CS as a discipline [1]. This description served as a basis for ACM's nationwide recommendations for undergraduate CS programs, prepared in 1968 [2]; one of the authors (PW) was among the primary writers of the 1968 report.

ACM's description of CS [1] identified effective transformation of information as a central concern:

> "Computer science is concerned with information in much the same sense that physics is concerned with energy... The computer scientist is interested in discovering the pragmatic means by which information can be transformed."

By viewing algorithms as transformations of input to output, ACM adapted an algorithmic approach to computation; this is made explicit in the next sentence of the report:

> "This interest leads to inquiry into effective ways to represent information, effective algorithms to transform information, effective languages with which to express algorithms... and effective ways to accomplish these at reasonable cost."

Having a central algorithmic concern, analogous to the concern with energy in physics, helped to establish CS as a legitimate academic discipline on a par with physics.

Knuth's famous and influential textbook, *The Art of Computer Programming, Vol. 1* [8] in the late 1960's popularized the centrality of algorithms prescribed by the ACM. In his discussion of an algorithm, Knuth was consistent with the mathematical function-based foundations of the Theory of Computation. He explicitly specified that algorithms are closed; no new input is accepted once the computation has begun:

> "An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins."

Knuth's careful discussion of algorithmic computation remains definitive to this day; in particular, it serves as the basis of the authors' understanding of this term.

Knuth distinguished algorithms from arbitrary computation that may involve I/O, ensuring their function-based behavior and guaranteeing their equivalence with TMs:

> "There are many other essentially equivalent ways to formulate the concept of an effective computational method (for example, using TMs)."

One example of a problem that is not algorithmic is the following instruction from a recipe [8]: "toss lightly until the mixture is crumbly." This problem is not algorithmic because it is impossible for a computer to know how long to mix; this may depend on conditions such as humidity

that cannot be predicted with certainty ahead of time. In the function-based mathematical worldview, all inputs must be specified at the start of the computation, preventing the kind of feedback that would be necessary to determine when it's time to stop mixing. The problem of driving home from work also illustrates the sort of problems that Knuth meant to exclude.

Algorithms, modeled by TMs, have remained central to computer science to this day. However, neither mathematics nor the ACM provided an explicit agreed-upon definition of an algorithm. As we will see, the inconsistencies between Knuth's definition and that of other early textbooks greatly contributed to the Turing Thesis myth.

### 3.2 Algorithms redefined

The 1960's decision by theorists and educators to place algorithms at the center of CS was clearly reflected in undergraduate textbooks. However, not everyone used this term in the same way. While some early textbooks such as [8] were careful to explicitly restrict algorithms to those that compute functions, and are therefore TM-equivalent, others left the restriction unstated.

An example is [10], one of the first textbooks on the theory of computation (whose later editions are being used to this day). The discussion of algorithms in [10] does not explicitly preclude non-functional computation, such as driving home from work:

> "A procedure is a finite sequence of instructions that can be mechanically carried out, such as a computer program... A procedure which always terminates is called an algorithm."

However, the prohibition against obtaining inputs dynamically during the computation is implicitly present. As already mentioned, ALGOL, the language then used for writing algorithmic programs, did not offer any constructs for input and output. The examples of various problems in [10] also make it clear that only function-based computation was considered.

Yet other textbooks, such as [11], explicitly broadened the idea of algorithms to include problems beyond those that can be solved by TMs. On the surface, the definition of an algorithm in [11] is no different from [10], leaving students with the impression that these computational problems could also be solved by TMs.

> "An algorithm is a recipe, a set of instructions or the specifications of a process for doing something. That something is usually solving a problem of some sort."

However, the examples of computable problems in [11] are no longer function based, admitting just the kind of computation that Knuth had rejected. Two such examples, that can supposedly be solved by an algorithm, are making potato vodka and filling a ditch with sand. The space of problems solvable by such "algorithms" is no longer the same as for Knuth; Turing's Thesis no longer applies here.

The subject of [11] was programming methodology rather than the Theory of Computation, and the mathematical principles that underpin our models of computation were cast aside for the sake of practicality. This approach is typical of non-theory textbooks, reflecting the centrality of algorithms without being restricted to the computation of functions. Rice made no claims of TM-equivalence for his "algorithms". However, the students were not made aware that this popular notion of algorithms is different from Knuth's, and that the set of problems considered computable had thereby been enlarged. [2]

[2] In private conversation with one of the authors (DG) in the fall of 1999, Knuth expressed some misgivings about his definition of an algorithm, and shared plans to broaden it if that text were ever rewritten. It is not clear what his plans were regarding the claim of equivalence between algorithms and TMs.

The undergraduate curriculum paired Rice's broader conceptualization of computation with theories claiming that every computable problem can be computed by TMs, creating the Turing Thesis myth.

# 4 What is a Turing Machine?

In this section, we discuss the remaining two claims of the Turing Thesis myth; they are both related to the limitations of early computers, when compared to the present day.

## 4.1 Syntax and semantics

According to the set-theoretic definition, a TM consists of a finite set of states, a read/write head, a tape, and a control mechanism for transitioning between states and performing read/write actions on the tape. At this level, the description of a TM is similar to that of a computer. The differences, as pointed out in [18], are that the computer's memory is not infinite, and it is accessed randomly rather than sequentially. But these differences are relatively minor, and the following claim has been made:

**Nature of computers:** TMs serve as a general model for computers.

This claim is one of the claims that make up the Turing Thesis myth.

Just as for any other class of automata such as FSA (finite state automata), the set-theoretic definition does not completely capture TMs; it captures their *syntax*, but not *semantics*. TMs are not fully specified until we define what is meant by their computation, i.e. their semantics. As defined by Turing [15], TM semantics prescribe that every computation starts in an identical configuration (except for the contents of the read/write tape), and the contents of the tape cannot be modified from the "outside" during the computation. This can be contrasted with Turing's alternative models of *choice machines* and *oracle machines*.

The complete TM definition includes both the set-theoretic definition and the semantic definition. If the semantics of the TM were defined differently, it may (or may not) produce an equivalent machine, but it would no longer be a TM. Statements about TM expressiveness, such as the Turing Thesis, fundamentally depend on their semantics.

Early computers did in fact compute as prescribed above. However, while perhaps reflecting TM syntax, the computation of modern computers is no longer based on the same semantics. Unlike TMs, new inputs arrive continuously (think of a document processor); the output is also produced continuously (in case of the document processor, it is the screen display of the document). There is I/O entanglement; later inputs are affected by earlier outputs and vise-versa. All this renders a computer's behavior non-functional; it no longer computes a function from the input to the output. We refer to it as *interactive computation*.

## 4.2 The Universal Turing Machine

A *Universal Turing Machine* is a special TM introduced by Turing [15], that can simulate any other TM. It served as the inspiration for the notion of *general-purpose computing*. Turing himself saw a direct parallel between the capability of a computer to accept and execute programs, and his notion of a universal machine.

The principle of universality can easily be extended to any other class of algorithmic machines. As long as each machine in that class can be captured by a finite description, prescribing what this machine would do in every possible configuration, a TM can be created to simulate all machines in that class:

**Universality Thesis:** Any class of effective devices for computing functions can be simulated by a TM.

Analogously to the Turing Thesis, the Universality Thesis combines with the mathematical worldview to obtain the following corollary:

**Universality Corollary:** Any computer can be simulated by a TM.

This corollary is the last of the four claims that make up the Turing Thesis myth.

Again, the undergraduate textbooks played a key role in using this claim to reinforce the Turing Thesis myth. In order to present the expressiveness of TMs in a more accessible (and dramatic) fashion, the Universality Corollary was melded with the Turing Thesis Corollary, resulting in the following statement that (incorrectly) summarized the role of TMs:

**Strong Turing Thesis:** A TM can do anything that a computer can do.

# 5 The present and future of computing

## 5.1 Modern theory textbooks

While the practical computer scientists have long since broadened the concept of algorithms beyond the computation of functions, theoretical computer science has retained the mathematical worldview. Despite advanced complexity theoretic work that ventures outside this worldview, such as on-line and distributed algorithms, Arthur-Merlin games, and interactive proofs, our treatment of the Theory of Computation at the undergraduate level has not changed. [3] Mathematical principles continue to frame computation as function-based, and to delimit our notion of a computational problem accordingly.

The popular book by Sipser [14] is an example of the resulting dichotomy. Sipser's discussion of algorithms resembles Rice's, but the equivalence with TMs is taken for granted:

> "an algorithm is a collection of simple instructions for carrying out some task. Commonplace in everyday life, algorithms sometimes are called procedures or recipes... The TM merely serves as a precise model for the definition of algorithm."

While Sipser's traditional selection of computational problems is all function-based, this description certainly leaves an impression that problems such as processing documents and driving home from work were not meant to be excluded. His claim of the Strong Turing Thesis, discussed earlier, reinforces that impression: [4]

> "A TM can do anything that a computer can do." [14]

While the Strong Turing Thesis is true when computers are limited to the task of computing functions or algorithms (as they were originally), it does not apply in the context of highly interactive computing systems of today and tomorrow, such as the internet, multimedia document processors, or robotic cars.

[3] A recent ACM SIGACT Newsletter acknowledges that of all undergraduate CS subjects, theoretical computer science has changed the least over the decades [13].

[4] In a 1999 letter to one of the authors (DG), Sipser acknowledged that there is a "difference in power" between interactive and algorithmic models of computation. However, he did not acknowledge that computers are interactive rather than algorithmic, and the implications thereof.

### 5.2 Time for new models

Hoare, Milner and others have long realized that TMs do not model all of computation [6]. However, when their theory of concurrent computation was first developed in the late '70s, it was premature to openly challenge TMs as a complete model of computation. Concurrency theory positions *interaction* as orthogonal to *computation*, rather than a part of it. By separating interaction from computation, the question whether the models for CCS and the $\pi$-calculus went beyond Turing machines and algorithms was avoided.

Researchers in other areas of theoretical computer science have also found need for interactive models of computation, such as Input/Output automata for distributed algorithms and Interactive TMs for interactive proofs. However, the issue of the expressiveness of interactive models vis-a-vis TMs was not raised until the mid-1990's, when the model of interaction machines as a more expressive extension of TMs was first proposed by one of the authors [5].

The theoretical framework for sequential interaction machines, as a persistent stream-based extension to TMs, was completed by the other author in [9]. This paper also presents an analog thesis to Turing's for the sequential interactive case. Van Leeuwen, a Dutch expert on the theory of computation, proposed an alternate extension in [17]. In addition to interaction, other ways to extend computation beyond Turing machines have been considered, such as quantum computing.

While not part of CS Theory, the field of AI has perhaps gone the furthest in explicitly recognizing the expressiveness gains of moving beyond algorithms. In the early 1990's, Rodney Brooks convincingly argued against the algorithmic approach of "good old-fashioned AI", positioning interaction is a prerequisite for intelligent system behavior [3]. This argument has been adopted by the mainstream AI community, whose leading textbooks recognize that interactive *agents* are a better model of intelligent behaviors than simple input/output functions [12].

However, the assumption that all of computation can be algorithmically specified is still widely accepted in the CS community, and interaction machines have been criticized as an unnecessary Kuhnian paradigm shift.

### 5.3 Common Myths Corrected

We have discussed the origins for the popularity of the Turing Thesis myth, having identified four distinct claims that comprise it:

**Claim 1.** (Mathematical worldview) All computable problems are function-based.

**Claim 2.** (Turing Thesis corollary) TMs can compute any solvable problem.

**Claim 3.** (Nature of computers) TMs serve as a general model for computers.

**Claim 4.** (Universality corollary) TMs can simulate any computer.

For each of these claims, there is a grain of truth. By reformulating them to bring out the hidden assumptions, misunderstandings are removed. The following versions of the above statements are correct:

**Corrected Claim 1.** All algorithmic computation is function-based.

**Corrected Claim 2.** TMs can compute any algorithmic problem.

**Corrected Claim 3.** TMs serve as a general model for early computers.

**Corrected Claim 4.** TMs can simulate any algorithmic computing device.

Furthermore, the following claim is also correct:

**Claim 5:** TMs cannot compute all problems, nor can they do everything that real computers can do.

This claim, while incompatible with original claims 1-4, is perfectly consistent with their corrected versions. It contradicts the Strong Turing Thesis, exposing the fallacy of the Turing Thesis myth. Dispelling this myth has grown more important as the practice of computing is becoming more and more interactive. Its algorithmic fundamental identity no longer serves it well.

In the last three decades, computing technology has shifted from mainframes and microstations to networked embedded and mobile devices, with the corresponding shift in applications from number crunching and data processing to the internet and ubiquitous computing. We believe it is no longer premature to encompass interaction as part of computation. A paradigm shift is necessary in our notion of computational problem solving, so it can better model the services provided by today's computing technology.

## References


[1] An Undergraduate Program in Computer Science-Preliminary Recommendations, A Report from the ACM Curriculum Committee on Computer Science. *Comm. ACM* 8(9), Sep. 1965, pp. 543-552.

[2] Curriculum 68: Recommendations for Academic Programs in Computer Science, A Report of the ACM Curriculum Committee on Computer Science. *Comm. ACM* 11(3), Mar. 1968, pp. 151-197.

[3] R. Brooks. Intelligence Without Reason. *MIT AI Lab Technical Report 1293.*

[4] M. Davis. *Computability & Unsolvability.* McGraw-Hill, 1958.

[5] P. Wegner. Why Interaction is More Powerful Than Algorithms. *Comm. ACM*, May 1997.

[6] P. Wegner, D. Goldin. Computation Beyond Turing Machines. *Comm. ACM*, Apr. 2003.

[7] P. Denning. The Field of Programmers Myth. *Comm. ACM*, July 2004.

[8] D. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms.* Addison-Wesley, 1968.

[9] D. Goldin, S. Smolka, P. Attie, E. Sonderegger. Turing Machines, Transition Systems, and Interaction. To appear in *Information & Computation J.*, 2004.

[10] J.E. Hopcroft, J.D. Ullman. *Formal languages and their relation to automata.* Addison-Wesley, 1969.

[11] J. K. Rice, J. N. Rice. *Computer Science: Problems, Algorithms, Languages, Information and Computers.* Holt, Rinehart and Winston, 1969.

[12] S. Russell, P. Norveig. *Artificial Intelligence: A Modern Approach.* Addison-Wesley, 1994.

[13] *SIGACT News*, ACM Press, March 2004, p. 49.

[14] M. Sipser. *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

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

[16] E. Eberbach, D. Goldin, P. Wegner. Turing's Ideas and Models of Computation. In *Alan Turing: Life and Legacy of a Great Thinker*, ed. Christof Teuscher, Springer 2004.

[17] J. v. Leeuwen, J. Wiedermann. The Turing Machine Paradigm in Contemporary Computing. in *Mathematics Unlimited - 2001 and Beyond*, eds. B. Enquist and W. Schmidt, Springer-Verlag, 2000.

[18] P. Wegner. *Programming Languages, Information Structures and Machine Organization*, McGraw-Hill, 1968.