



## Second Thoughts about Church's Thesis and Mathematical Proofs

Elliott Mendelson

*The Journal of Philosophy*, Vol. 87, No. 5. (May, 1990), pp. 225-233.

Stable URL:

<http://links.jstor.org/sici?sici=0022-362X%28199005%2987%3A5%3C225%3ASTACTA%3E2.0.CO%3B2-G>

*The Journal of Philosophy* is currently published by Journal of Philosophy, Inc..

---

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at <http://www.jstor.org/about/terms.html>. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at <http://www.jstor.org/journals/jphil.html>.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

---

JSTOR is an independent not-for-profit organization dedicated to and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact [support@jstor.org](mailto:support@jstor.org).

---

---

# THE JOURNAL OF PHILOSOPHY

VOLUME LXXXVII, NO. 5, MAY 1990

---

---

## SECOND THOUGHTS ABOUT CHURCH'S THESIS AND MATHEMATICAL PROOFS

I intend to renounce the standard views concerning the nature of Church's thesis. In order to put things in proper perspective, I shall give a brief introduction to the relevant notions and a resumé of the history of the thesis.

The central notion is that of *algorithm*. An algorithm is an effective and completely specified procedure for solving a whole class of problems. Examples are the well-known algorithms for adding and multiplying any two natural numbers. In logic, the truth-table technique of testing a statement form to see whether it is a tautology is an algorithm. An algorithm does not require ingenuity; its application is prescribed in advance and does not depend upon any empirical or random factors.

An algorithm that is applied to natural numbers and yields a natural number as a value is called an *effectively computable function*. In general, the objects that an algorithm is applied to and the objects that it yields can be represented by natural numbers. Therefore, for the sake of simplicity, it is customary to confine attention to effectively computable functions.

It is crucial to understand certain things about effectively computable functions. First of all, we do not mean actual human computability or empirically feasible computability. A function can be computable even though the instructions for it may require more symbols than there are atoms in the universe. Likewise, a value of an effectively computable function may require more time for its computation than the entire past and future history of the human race. When we talk about computability, we ignore any limitations of space, time, or resources. Second, functions are treated extensionally. A function is determined by the ordered pairs of its arguments

and values, not by a particular way in which it is defined. For example, consider the following function:

$$f(x) = \begin{cases} 1 & \text{if Fermat's Last Theorem is true} \\ 0 & \text{if Fermat's Last Theorem is false} \end{cases}$$

This function is an effectively computable function. It is either the constant function 1 (which is effectively computable) or it is the constant function 0 (which is also effectively computable). At the present time, we happen not to know how to compute any values of  $f(x)$ . Thus, we are adopting a completely classical, nonintuitionistic stance.<sup>1</sup>

Many concrete algorithms have long been known in the history of mathematics and logic. In all such cases, the fact that the alleged algorithm actually is an algorithm is an intuitively obvious fact. (For example, it was so easy to see that our method of adding two natural numbers is an algorithm that we never bothered to prove it. Therefore, it never entered our mind that the notion of algorithm required any further elucidation.) This was not felt to be a difficulty until mathematicians and logicians began to encounter important classes of problems for which there seemed to be no algorithmic solutions. (Examples: (a) Is a given first-order formula logically valid? (b) Is a given arithmetic statement derivable from the axioms of Peano arithmetic? (c) Is a given mathematical statement true? (d) Is a given equation true in a given finitely presented group?) To show the nonexistence of algorithms for solving these problems seemed to call for a mathematically precise equivalent of the notions of algorithm or effectively computable function. The "pressure" of these problems finally became so great that in the middle of the 1930s several quite different explications were independently proposed for the notion of effectively computable function. These proposals were all published in the same year, 1936, by Alonzo Church, Emil Post, and Alan Turing.<sup>2</sup> It turned out that they (and many other later ones) are mutually equivalent. Let us look at Turing's approach, which is the most convincing.

<sup>1</sup> It was from this classical viewpoint that Church's thesis was originally proposed and discussed, and it is still commonly used with such an interpretation. By keeping to this classical setting, I am not denying the tenability of a constructivist philosophy of mathematics.

<sup>2</sup> Church, "An Unsolvable Problem of Elementary Number Theory," *Amer. J. of Mathematics*, LVIII (1936): 345-363; Post, "Finite Combinatory Processes. Formulation I," *Journal of Symbolic Logic*, 1 (1936): 103-5; Turing, "On Computable Numbers with Applications to the *Entscheidungsproblem*," *Proceedings of the London Mathematical Society*, XLII (1936): 230-265.

Turing wanted to capture the essence of computability by setting up mathematical "models" that would generate all functions, and only such functions, that are computable. These models are what we now call *Turing machines*. When calculating we require a field of computation, such as a piece of paper or a blackboard. A Turing machine uses for this purpose a tape, divided into squares. At any one moment, this tape consists of finitely many squares, but, when necessary, additional squares can be added to the right and left. At any given moment, the squares of the tape may be occupied by certain symbols, at most one symbol to a square. These symbols are chosen from a finite set of symbols, the *alphabet* of the machine. The Turing machine has a *reading device* or *scanner* which, at any given moment, is observing the content of one of the squares of the tape. At any one moment, the scanner can be in any one of a fixed finite collection of *internal states*. (These states correspond to the presumably finite number of states that characterize the structure of a person or computing machine at any given time.) The Turing machine also has a *program*, a finite list of instructions that tell it what to do under all circumstances. If the scanner is in a certain state and is observing a certain symbol, these instructions tell it to do one of four things: (1) replace the observed symbol by another (possibly the same) specific symbol; (2) move one square to the right; (3) move one square to the left; (4) stop. After the appropriate action is performed, the new internal state of the scanner is determined by the program. A Turing machine computes a function  $f(x_1, \dots, x_n)$  of  $n$  arguments in the following manner. At the beginning of the computation, the  $n$  arguments are written on the tape, separated by blank squares. (Each argument, a natural number  $m$ , is represented by a sequence of  $m + 1$  strokes,  $||| \dots |$ , written in consecutive squares.) The machine starts off in a special state, called the *initial state*, and proceeds to operate according to its instructions. If the machine stops and the only thing left on the tape is the representation of a certain number  $k$ , then  $k$  is taken to be the value  $f(x_1, \dots, x_n)$ . A function computable by some Turing machine is called a *Turing-computable function*. Such a function can be *partial*, that is, it need not be defined for all  $n$ -tuples.

Notice that Turing made several normalizing assumptions. Although we usually use at least a two-dimensional field of computation, he restricts the computation to a one-dimensional tape. This really is not an essential limitation, since it is well-known how to encode higher dimensional arrays of symbols in terms of a string of symbols. Second, although we usually move freely around our field of computation, Turing restricts his machine to movements to an

adjacent square. Here, one has to verify that this is not a serious limitation; the effect of a "jump" from a square to a nonadjacent square can be accomplished by a suitable sequence of movements to adjacent squares. Third, the fact that the alphabet is finite is not a problem. Actual computing systems use finite alphabets; moreover, a denumerable alphabet  $a_1, a_2, \dots$  can be replaced by a two-symbol alphabet  $\{a, '\}$  by letting  $a'$  stand for  $a_1$ ,  $a''$  for  $a_2$ , and so on. Fourth, the restriction to a finite set of internal states seems to be acceptable. Turing justifies this as follows: "If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused." (If there is any weak link in Turing's presentation, this is probably it.) Church's thesis can now be formulated.

Church's thesis (CT): A function is effectively computable if and only if it is Turing-computable.

The Turing-computable functions have been shown to be identical with the so-called *partial recursive functions* (defined by S. C. Kleene<sup>3</sup>). Church's thesis is often stated in terms of partial recursive functions instead of Turing-computable functions, because the former are easier to work with from a technical standpoint.

Here are some of the arguments in favor of CT.

- (a) The host of notions proposed as equivalents of effectively computable function have all turned out to be equivalent. This seems to show that the underlying intuitive target notion has been hit.<sup>4</sup>
- (b) It is easy to see that one half of CT is true: every Turing-computable function is effectively computable.
- (c) The other half of CT, that every effectively computable function is Turing-computable, has been confirmed for the very large number of cases in which it has been tested.
- (d) The arguments given by Turing show that the special methods of computation used by Turing machines involve no essential restriction on the general methods of computation employed by (idealized) human beings or computers.

CT is now almost universally accepted by mathematicians and logicians. In the theory of recursive functions, it is constantly resorted to without explicit mention.<sup>5</sup> Moreover, all the limitative theorems of modern logic, such as Gödel's incompleteness theorem, depend for their significance on the validity of CT. For example, Church's theorem states that there is no recursive decision procedure for

<sup>3</sup> *Introduction to Metamathematics* (Princeton: Van Nostrand, 1952).

<sup>4</sup> Of course, this is not conclusive. It is conceivable that all the equivalent notions define a concept that is related to, but not identical with, effective computability.

<sup>5</sup> See the standard reference work by Hartley Rogers, Jr., *Theory of Recursive Functions and Effective Computability* (New York: McGraw-Hill, 1967).

first-order logical validity.<sup>6</sup> It is CT that then allows us to draw the important conclusion that there is no effective procedure for deciding whether a first-order formula is logically valid.

CT is generally regarded as an assertion of the extensional equivalence of the notions of effectively computable function and partial recursive function (or Turing-computable function). Because the former notion is intuitive and the latter is alleged to be a precise mathematical concept, it is further asserted that it is impossible to *prove* CT; hence, CT is just a conjecture or a thesis, and is not susceptible of a rigorous proof. This is what almost everybody believes about CT; for example, Church and Kleene:

This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion (Church, *op. cit.*, §7).

Since our original notion of effective calculability of a function is a somewhat vague intuitive one, [CT] cannot be proved . . . While we cannot prove [CT], since its role is to delimit precisely a hitherto vaguely conceived totality, we require evidence that it cannot conflict with the intuitive notion which it is supposed to complete; i.e. we require evidence that every particular function which our intuitive notion would authenticate as effectively calculable is [recursive]. The thesis may be considered a hypothesis about the intuitive notion of effective calculability, or a mathematical definition of effective calculability; in the latter case, the evidence is required to give the theory based on the definition the intended significance (Kleene, *op. cit.*, pp. 317-9).

These standard views about CT hold it to be a *rational reconstruction*, in the sense of Rudolf Carnap and Carl Hempel. A rational reconstruction is a precise, scientific concept that is offered as an equivalent of a prescientific, intuitive, imprecise notion. In all standard cases in which the original intuitive notion is definitely known to apply or not to apply, the rational reconstruction should yield the same outcome. The rational reconstruction may go beyond the original notion, however, in cases where the latter notion is not determinate. Moreover, the rational reconstruction need not, and usually does not, have the same psychological effect as the original notion; expert users of the language generally do not think of the rational reconstruction when they use the original notion. Confirmation of the correctness of a rational reconstruction apparently must involve, at least in part, an empirical investigation. The correctness of the reconstruction cannot be proved.

<sup>6</sup> More precisely, if  $f$  is the function such that  $f(n) = 1$  if  $n$  is the Gödel number of a logically valid first-order formula and  $f(n) = 0$  otherwise, then  $f$  is not recursive.