

The Theses of Church and Turing

Many logicians and mathematicians have attempted to characterize the computable functions by defining systems in which computation could be carried out, or at least, described. Some of the historical highlights in the formulation of computation were due to Aristotle, Euclid, Frege, Hilbert, and Russell and Whitehead. Some of these mathematicians developed systems for fragments of arithmetic or geometry, but there were always problems and thus none could produce a system in which all of human computation could be formulated properly.

The mid-nineteen-thirties brought Alonzo Church and Alan Turing to the attention of mathematicians throughout the world. Both had developed very different systems in which computation could be carried out and each of them felt that any computation which was humanly or mechanically possible could be carried out within the systems of computation which they formulated. They also felt that under certain common sense rules, no systems of computation could be more powerful than theirs. In other words, they believed (as many others had before) that they had found an answer to a question that had eluded scholars for over two thousand years!

These beliefs have become known as *Church's Thesis* and *Turing's Thesis*. We must note that *neither belief can be proven*, and this is not too difficult to see. After all, how can one prove that all things which are computable can be done by Turing machines unless one somehow lists all of these things and then shows how to compute them. It is the same with systems. How do we know that someone will not propose a system for computation next week that is more powerful than Turing machines?

But, almost everyone *does* believe Church and Turing. This is because lots of evidence for these two beliefs has been presented to date. We shall delve into this after we have precisely formulated the theses of Church and Turing.

First, we need to explore just what is meant by human computation. Let us look closely at programs and machines. They share two major features. First, all computations are ***finitely specified***. This means that the set of instructions followed to carry out a computation must be finite. Exactly like a recipe or a program.

The second shared principle is a little more complicated. We note that all computations carried out in these systems are ***effective*** and ***constructive***. By effective we mean actual or real. No imaginary computations are allowed. By

constructive we mean that it must be possible to show exactly how the computation is performed. We need to be able to reveal the actual sequence of computational steps.

Let us resort to a few examples. Functions we often compute such as:

$$x + y \quad \text{or} \quad x \leq y + 3$$

are constructive because we know exactly how to compute them using simple arithmetic or logical algorithms. We know all of the steps needed to *exactly construct* the answer. We could even design and build an actual computer chip to do the job. There is absolutely no question at all about it. Something a little more difficult like:

$$f(n) = \text{the } n^{\text{th}} \text{ digit of } \pi$$

can be computed by an effective, constructive algorithm. This has been a well documented mathematical avocation for years. There is also a constructive algorithm for converting this text to postscript, pdf, or html.

Even some of the partial functions we have seen before are constructive and effective. Consider the prime number recognizer:

$$p(x) = \begin{cases} x & \text{if } x \text{ is a prime integer} \\ \text{diverge} & \text{otherwise} \end{cases}$$

We know exactly how to design a program that performs this task. An inefficient method might involve checking to see if any integer less than \sqrt{x} divides x evenly. If so, then the routine halts and presents x as the output, otherwise it enters an infinite loop.

But consider the following Boolean function.

$$f(x, y) = \text{if lightning strikes at latitude } x \text{ and longitude } y$$

Is this computable? We feel that it is not since one must wait forever on the spot and observe in order to determine the answer. We cannot think of any effective and constructive way to compute this.

Other popular functions that many wish were constructive are:

$$\begin{aligned} w(n) &= \text{the } n^{\text{th}} \text{ number from now that will come up on a roulette wheel} \\ h(n) &= \text{the horse that will win tomorrow's } n^{\text{th}} \text{ race} \end{aligned}$$

We can statistically try to predict the first, but have no effective way of computing it. The latter is totally out of the question. If these functions were effective and constructive then computer programmers would be millionaires!

With our definition of human computation in hand we shall state the first of the two beliefs.

Church's Thesis: *Every finitely specified, constructive computing procedure can be carried out by a Turing machine.*

By the way, Church did not state the thesis in terms of Turing machines. He stated it in terms of the lambda calculus.

Note that anyone who subscribes to Church's thesis no longer needs to design Turing machines! This is wonderful news. It means that we need not write down lengthy sequences of machine instructions in order to show that something is computable if we can state the algorithm in a more intuitive (but constructive) manner. Part of the justification for this is that *each and every* finitely specified, constructive algorithm anyone has ever thought up has turned out to be Turing machine computable. We shall appeal to Church's thesis often in the sequel so that we may omit coding actual machine descriptions.

Our second belief is credited to Turing and deals with systems of computation rather than individual functions.

Turing's Thesis: *Any formal system in which computation is defined as a sequence of constructive steps is no more powerful than the Turing machine model of computation.*

In the section dealing with machine enhancement we saw some material which could be thought of as evidence for Turing's thesis. And history has not contradicted this thesis. Every formal system of the type specified above which anyone has ever invented has been shown to be no more than equivalent to Turing machines. Some of these include Church's *lambda calculus*, Post's *Post machines*, Markov's *processes*, and Herbrand-Gödel-Kleene *general recursive functions*.

So, it seems that we have achieved our goal and defined computation. Now it is time to examine it.