und.1 Introduction

tur:und:int:sec

It might seem obvious that not every function, even every arithmetical function, can be computable. There are just too many, whose behavior is too complicated. Functions defined from the decay of radioactive particles, for instance, or other chaotic or random behavior. Suppose we start counting 1-second intervals from a given time, and define the function $f(n)$ as the number of particles in the universe that decay in the n-th 1-second interval after that initial moment. This seems like a candidate for a function we cannot ever hope to compute.

But it is one thing to not be able to imagine how one would compute such functions, and quite another to actually prove that they are uncomputable. In fact, even functions that seem hopelessly complicated may, in an abstract sense, be computable. For instance, suppose the universe is finite in time—some day, in the very distant future the universe will contract into a single point, as some cosmological theories predict. Then there is only a finite (but incredibly large) number of seconds from that initial moment for which $f(n)$ is defined. And any function which is defined for only finitely many inputs is computable: we could list the outputs in one big table, or code it in one very big Turing machine state transition diagram.

We are often interested in special cases of functions whose values give the answers to yes/no questions. For instance, the question "is n a prime number?" is associated with the function

$$\text{isprime}(n) = \begin{cases} 1 & \text{if } n \text{ is prime} \\ 0 & \text{otherwise.} \end{cases}$$

We say that a yes/no question can be *effectively decided*, if the associated 1/0-valued function is effectively computable.

To prove mathematically that there are functions which cannot be effectively computed, or problems that cannot effectively decided, it is essential to fix a specific model of computation, and show about it that there are functions it cannot compute or problems it cannot decide. We can show, for instance, that not every function can be computed by Turing machines, and not every problem can be decided by Turing machines. We can then appeal to the Church-Turing thesis to conclude that not only are Turing machines not powerful enough to compute every function, but no effective procedure can.

The key to proving such negative results is the fact that we can assign numbers to Turing machines themselves. The easiest way to do this is to enumerate them, perhaps by fixing a specific way to write down Turing machines and their programs, and then listing them in a systematic fashion. Once we see that this can be done, then the existence of Turing-uncomputable functions follows by simple cardinality considerations: the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (in fact, even just from $\mathbb{N}$ to $\{0, 1\}$) are non-enumerable, but since we can enumerate all the Turing machines, the set of Turing-computable functions is only denumerable.

We can also define *specific* functions and problems which we can prove to be uncomputable and undecidable, respectively. One such problem is the so-called *Halting Problem.* Turing machines can be finitely described by listing their instructions. Such a description of a Turing machine, i.e., a Turing machine program, can of course be used as input to another Turing machine. So we can consider Turing machines that decide questions about other Turing machines. One particularly interesting question is this: "Does the given Turing machine eventually halt when started on input n?" It would be nice if there were a Turing machine that could decide this question: think of it as a quality-control Turing machine which ensures that Turing machines don't get caught in infinite loops and such. The interestign fact, which Turing proved, is that there cannot be such a Turing machine. There cannot be a single Turing machine which, when started on input consisting of a description of a Turing machine M and some number n, will always halt with either output 1 or 0 according to whether M machine would have halted when started on input n or not.

Once we have examples of specific undecidable problems we can use them to show that other problems are undecidable, too. For instance, one celebrated undecidable problem is the question, "Is the first-order formula φ valid?". There is no Turing machine which, given as input a first-order formula φ, is guaranteed to halt with output 1 or 0 according to whether φ is valid or not. Historically, the question of finding a procedure to effectively solve this problem was called simply "the" decision problem; and so we say that the decision problem is unsolvable. Turing and Church proved this result independently at around the same time, so it is also called the Church-Turing Theorem.

Photo Credits

Bibliography