Incompleteness via the halting problem

Jeremy Avigad

February 21, 2005

On his home page, Haim Gaifman has posted a note that explains how one can understand the Gödel sentence in terms of Cantor's diagonal argument. Here, I explain how it can be understood in terms of the undecidability of the halting problem.

1 The halting problem

Let $f_m(x)$ denote the partial function from $\mathbb{N}$ to $\mathbb{N}$ computed by the Turing machine coded by the natural number m, under some reasonable encoding of Turing machines. Then the sequence $f_0, f_1, f_2, \ldots$ enumerates the unary partial computable functions, and, moreover, uniformly, in the sense that the function $g(i, x) \simeq f_i(x)$ is computable.

There is no such enumeration of the unary total computable functions. To see this, suppose $k_0, k_1, k_2 \ldots$ is an enumeration of computable functions such that the function $l(i, x) = k_i(x)$ is computable. Then the function $d(x) = l(x, x) + 1 = k_x(x) + 1$ is also computable. But d cannot be any function k_i, since for each i, $d(i) = k_i(i) + 1$.

(Alternatively, if we define $m(x) = \max\{k_0(x), k_1(x), \ldots, k_x(x)\} + 1$, we obtain a function m with the stronger property that it "eventually dominates" each function k_i. To see this, note that for every i, $m(x)$ is greater than $k_i(x)$ for every $x \geq i$.)

Let $h(m, x)$ be defined by

$$h(m, x) = \begin{cases} 1 & \text{if } f_m(x) \text{ is defined} \\ 0 & \text{otherwise} \end{cases}$$

The function $h(m, x)$ is a version of the "halting problem," since it amounts to a decision, for each m and x, as to whether Turing machine m halts on input x.

The simple diagonalization above gives us an indirect way of seeing that the halting problem is unsolvable, i.e. that h is not computable. For, if h were computable, we could compute

$$l(m, x) = \begin{cases} f_m(x) & \text{if } h(m, x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Defining $k_i(x) = l(i, x)$ would then yield an enumeration of all the total computable functions, and we have showed that this is impossible.

One can prove that h is not computable more directly. Suppose h were computable. Then we could define a a partial function $s(x)$ by

$$s(x) \simeq \begin{cases} 0 & \text{if } h(x, x) = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

By our hypothesis, $s(x)$ is a partial computable function. Let m be such that $s = f_m$. Then $s(m)$ is defined, by definition, if and only if $h(m, m) = 0$; but by the definition of h, this happens if and only if $f_m(m)$, i.e. $s(m)$, is undefined.

2 Provably total computable functions

In these notes, we will be interested in effectively axiomatized theories of "interpret a reasonable amount of arithmetic." In particular, if a consistent theory T interprets Robinson's theory Q, then it can represent the notion

> s is a halting computation sequence for Turing machine m starting with input x

in such a way that for every m, x, and s, s really is a halting computation sequence for machine m on input x if and only if T proves that this is the case.

Such a theory T is said to be Π_2-*sound* if the following holds:

> For every m, T proves "for every x, machine m halts on input x" if and only if for every x, machine m halts on input x.

A computable function f from the natural numbers to the natural numbers is said to be *provably total* in a theory T if there is some machine, m, computing f such that T proves "for every x, machine m halts on input x." Saying that T is Π_2-sound simply amounts to the assertion that whenever T proves a statement of this form, then machine m really computes a total function.

There is a natural way of enumerating all the provably total computable functions of such a theory: for each i, if i describes a proof of the statement "for every x, machine m halts on input x" for some i, let k_i be the function computed by machine m; otherwise, let k_i be the constant zero function.

Assuming T is effectively axiomatized, this enumeration is uniform, in the sense that for every i and x we can compute $l(i, x) = k_i(x)$. But we know from the preceeding section that the sequence $k_0, k_1, k_2, \ldots$ cannot enumerate *all* the computable functions. This means that there must be a computable function s that is not on the list. If m is any Turing machine that computes s, then m halts on every input, but T does not prove that m halts on every input. Thus we have shown that there is a true sentence that cannot be proved in T.

In fact, Gödel's notion of *ω-consistency* implies Π_2-soundness. For the Turing machine m described in the preceding paragraph, it also implies that T does not prove (falsely) that m *doesn't* halt on some input, since for each particular input, T can verify that machine m really does halt. Thus we have shown:

Theorem 2.1 *Let T be an ω-consistent, effectively axiomatized theory interpreting Q. Then there is a sentence φ that is neither provable nor refutable in T.*

This is a weak version of first incompleteness theorem.

It is instructive to spell out the argument in greater detail. For each i, let k_i be the ith provably total computable function in T. Let $s(x) = k_x(x) + 1$. We have argued that:

1. $s(x)$ is a total computable function (since T is effectively axiomatized, and Π_2-sound), but
2. T doesn't prove $s(x)$ is total (by the diagonal argument).

Why is it that *we* can prove that $s(x)$ is total, but T can't? The answer is that we have made central use of the fact that T is Π_2-sound. With slightly stronger assumptions on the amount of arithmetic one can develop in T, this argument can formalized. Thus, we have shown:

Theorem 2.2 *Let T be as above, and assume T interprets a sufficient amount of arithmetic. Then T does not prove its Π_2-soundness.*

This is a weak version of the second incompleteness theorem.

3 The incompleteness theorems

We can obtain better versions of the incompleteness theorems using the more direct proof of the unsolvability of the halting problem described in Section 1. Given an effectively axiomatized theory T, consider the following attempt to solve the halting problem: on input m and x, simultaneously do the following:

1. simulate Turing machine m on input x, to see if it halts; and
2. search for a proof in T that m *doesn't* halt on input x.

In the first case, output 1 ("yes"), and in the second case, output 0 ("no").

Clearly any "yes" answer is a correct answer to the halting problem. Assuming T is consistent, every "no" answer has to be correct as well. But we know that the halting problem is unsolvable, which means that something has to go wrong. The only thing that can possibly go wrong is that there may be a Turing machine m and an input x, such that m does not halt on input x, but T does not prove this fact. The unsolvability of the halting problem implies that this is, indeed, the case.

On the other hand, assuming T is ω-consistent, for m and x as above, T cannot prove (falsely) that m *does* halt on input x, since for each particular s it can verify that s is not a halting computation sequence. Letting φ be the sentence "machine m does not halt on input x," we have shown the following:

Theorem 3.1 *Let T be an effectively axiomatized theory interpreting Q. Then there is an assertion φ such that if T is consistent, it does not prove φ, and if T is ω-consistent, it does not prove $\neg\varphi$.*

This is essentially Gödel's version of the first incompleteness theorem. It is stronger than Theorem 2.1 since for one part of the independence we only need to assume the consistency of T.

Again, it is instructive to unwrap the argument and make it more direct. For each x, let $\psi(x)$ be the assertion that Turing machine x does not halt on input x. Let m be a Turing machine that, on input x, searches for a proof of $\psi(x)$ in T, and halts if it finds one.

Assuming T is consistent, it does not prove $\psi(m)$, that is, the assertion that machine m does not halt on input m. For, if there is such a proof, then machine m *does* halt on input m, and this fact is also provable in T, making T inconsistent.

Assuming T is ω-consistent, it does not prove $\neg\psi(m)$, that is, the assertion that m *does* halt on input m. For, we have just shown that m does not halt on input m, and T can therefore verify, for each s, that s is not a halting computation sequence.

The statement $\psi(m)$, that is, the assertion that m does not halt on input m, is a version of the Gödel sentence. The original Gödel sentence asserted "I am not provable"; this one asserts that "the algorithm searching for a proof of me doesn't halt." The only difference is that the first is cast in terms of proofs, and the second is cast in terms of algorithms. Both arguments rely on the relationship between provability and computability.

Note that we have described a Turing machine m such that

1. m does not halt on input m, but
2. assuming T is consistent, T does not prove that m does not halt on input m.

With appropriate restrictions on T, T can carry out these arguments. Since it can also show that the conlusion of T implies 1, we have:

Theorem 3.2 *Assuming T is consistent, effectively axiomatized, and interprets a sufficient amount of arithmetic, T does not prove its own consistency.*

This is the second incompleteness theorem.

One can tell a similar story to explain Rosser's strengthening of the first incompleteness theorem, which lifts the assumption of ω-consistency.