Chapter 8

The Kleene Recursion Theorem

Pill Grates at Macrohard Corporation hates the numbering $\phi_{\_}$. He wants, of course, to replace it with his competing, proprietary numbering $\eta_{\_}$ that is subtly different in useless ways. Since Pill knows that $\eta_{\_}$ isn't very good, he has to resort to industrial cunning in order to sell it. His plan is to wipe $\phi_{\_}$ off of the planet by releasing MH-Virus$^{\text{TM}}$into the ambient computing environment. The Coke Classic$^{\text{TM}}$swilling MH-Virus$^{\text{TM}}$design team has been ordered to find a computer virus that wrecks the performance of every program index in the $\phi_{\_}$ system. That is, MH-Virus$^{\text{TM}}$is supposed to compute a total recursive function v such that for each n, k,

$$\phi_n^k \neq \phi_{v(n)}^k.$$

You are agent Gödel number 007 and your job is to foil Grates' diabolical plan. Your first question is, naturally, one of *principle*. Can a plan that diabolical possibly succeed, or is Grates just wasting his money?

Here is an analogy from analysis. Suppose you are challenged to drag a pencil from the left side to the right side of a square of paper without crossing the diagonal running from lower left to upper right and without lifting the pencil. You'll find you can't do it, because the diagonal is "in the way" (take a moment to try it). In mathematical terminology, the unbroken pencil trail is a continuous mapping f of the closed unit interval $[0, 1]$ into itself. If the pencil line crosses the diagonal, that means that some pair $(x, x) \in f$, so $f(x) = x$. So no continuous mapping of the closed unit interval into itself alters every point. A point unaltered by f is called a **fixed point** of f, so we can summarize the discussion by saying that every continuous mapping of the closed interval into itself has a fixed point, or a point it leaves unaltered.

Pill Grates wants a computable deformation of the partial recursive functions that alters each function. You might now suspect that his plan faces a similar difficulty: that some program's behavior must somehow end up unaltered. That is precisely what happens. In fact, the construction is literally analogous to

showing that each unbroken pencil line across a square must touch the diagonal of the square.

In the statement of the following theorem, total recursive f represents Grates' MH-Virus$^{\mathrm{TM}}$and the theorem says that there is some index whose input-output behavior isn't altered by f. That isn't to say that the computations themselves are unaltered— f may succeed in slowing down all computations to an impractical degree, for example. And the index whose behavior is preserved may be very uninteresting. It may be an index for the everywhere undefined function, for example.

The Kleene Recursion Theorem

$$(\forall k)(\forall f \in Tot)(\exists n)[(\phi_n^k = \phi_{f(n)}^k)].$$

Proof. I'll do it for the case of $k = 1$ and drop the annoying superscripts. The proof of this theorem is facilitated by a convention. Consider the embedded expression

$$\phi_{\phi_n(\vec{x})}(\vec{y}).$$

It is pretty clear that if

$$\phi_n(\vec{x}) \simeq z$$

then

$$\phi_{\phi_n(\vec{x})} = \phi_z.$$

But what if $\phi_n(\vec{x})$ is undefined? Then our numbering isn't "given a number to interpret", so $\phi_{\phi_n(\vec{x})}(\vec{y})$ does not denote a function. But there is another way to look at it. Let u be a universal index for numbering $\phi_{\_}$. Now we have that for each y,

$$\phi_u(\phi_n(\vec{x}), \langle \vec{y} \rangle) \uparrow,$$

where $\uparrow$ means "undefined". Thus, we may *also* think of the whole expression as denoting the everywhere undefined function

$$\phi_{\phi_n(\vec{x})} = \emptyset.$$

Now let an arbitrary, total recursive "virus" f be given. Think of a two-dimensional table in which the cell $T[n, m]$ is filled by the function $\phi_{\phi_n(m)}$. The table looks like:

$$\begin{array}{cccc}
\underline{\phi_{\phi_0(0)}} & \phi_{\phi_0(1)} & \phi_{\phi_0(2)} & \cdots \\
\phi_{\phi_1(0)} & \underline{\phi_{\phi_1(1)}} & \phi_{\phi_1(2)} & \cdots \\
\phi_{\phi_2(0)} & \phi_{\phi_2(1)} & \underline{\phi_{\phi_2(2)}} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{array}$$

(Think of this table as standing in for the closed unit interval of reals in our analogy). Every cell of the table is filled with a partial recursive function

because of our convention for dealing with the case in which $\phi_n(m)$ is undefined. Consider the (underlined) diagonal of the table. We will now see, remarkably enough, that the diagonal of the table

$$\phi_{\phi_0(0)}, \phi_{\phi_1(1)}, \phi_{\phi_2(2)}, \cdots$$

is also a row of the table

$$\phi_{\phi_z(0)}, \phi_{\phi_z(1)}, \phi_{\phi_z(2)}, \cdots$$

where the function ϕ_z generating the row is a total recursive function. We do this using the universal and *s-m-n* properties as follows. Using a universal index u of numbering $\phi^{\_}_{\_}$, define partial recursive function

$$\begin{aligned} \psi(n,x) &\simeq \phi_u(\phi_n(n), \langle x \rangle) \\ &\simeq \phi_{\phi_n(n)}(x). \end{aligned}$$

Since $\phi^{\_}_{\_}$ is onto $Part$, choose w such that

$$\phi_w(n,x) \simeq \psi(n,x).$$

Using the *s-m-n* property of numbering $\phi^{\_}_{\_}$, we obtain a total recursive s such that

$$\phi_{s(w,n)}(x) \simeq \phi_w(n,x).$$

Now compose in a constant function to obtain a unary total recursive g such that for all n,

$$g(n) = s(w,n).$$

Let $\phi_j = g$. Unwinding the definitions, we obtain:

$$\begin{aligned} \phi_{\phi_j(n)}(x) &\simeq \phi_{g(n)}(x) \\ &\simeq \phi_{s(w,n)}(x) \\ &\simeq \phi_w(n,x) \\ &\simeq \psi(n,x) \\ &\simeq \phi_u(\phi_n(n), \langle x \rangle) \\ &\simeq \phi_{\phi_n(n)}(x). \end{aligned}$$

So as promised, the diagonal of the table is also the jth row table where $\phi_j = g$ is a total recursive function:

$$\phi_{\phi_j(n)} = \phi_{g(n)} = \phi_{\phi_n(n)}.$$

Now consider the total recursive virus f. Since g is total recursive, so is $\phi_n = C(f,g)$. Let so we also have that

$$\phi_{f(g(0))}, \phi_{f(g(1))}, \phi_{f(g(2))}, \cdots.$$

is the nth row of the table. Now (just as in our attempt to draw a line across the unit square), this row intersects the diagonal at $\phi_{\phi_n(n)} = \phi_{f(g(n))}$. Now let's check the effect of the virus f on the index $g(n)$, which exists because g is total:

$$\begin{aligned} \phi_{f(g(n))}(x) &\simeq \phi_{\phi_n(n)}(x) \\ &\simeq \phi_{g(n)}(x). \end{aligned}$$

So the behavior of the index $g(n)$ is unaltered by f. $\dashv$

Exercise 8.1 *Earn a middle-management position at Macrohard: design a total recursive virus f such that the fixed point index guaranteed by Kleene's theorem has to compute the everywhere undefined function $\emptyset$. What does this tell you about the possibility of proving a fixed point theorem for Prim?*

Exercise 8.2 *Your mean boss at Macrohard saw what you just wrote in the previous exercise. To keep your job, you now have to write a virus that alters the behavior of every index that computes a non-total function. Write one and find a total function index whose input-output behavior isn't changed! Intuitively, why is it hard to write a virus that wrecks both total and partial programs?*

8.1 Where Did it Come From?

The proof of the recursion theorem is easy to follow, but not to come up with. How did Kleene even formulate the theorem prior to proving it?

Kleene was a student of Alonzo Church, whose pet invention was the **lambda calculus**. The idea is really fairly simple. Strictly speaking, the function f is denoted by the logical constant f, not by the open formula $f(x)$. But while it is easy to say that + denotes addition and *sqrt* denotes the square root function, sooner or later we run out of primitive symbols, as with polyomial functions like $2x^2 + x$, so we want to say "the function $f(x) = 2x^2 + x$". Of course, we "know what we mean", but this is technically "notational abuse", a felony in seven southern states.

Wouldn't it be nice if there were a notational device to distinguish the open formula from the function it defines? That is what the lambda calculus provides. Read

$$(\lambda x)[2x^2 + x]$$

as "the function of x whose value on argument x is $2x^2 + 7x$". Then you can write

$$((\lambda x)[2x^2 + x])(2) = 2 \cdot 2^2 + 2 = 10.$$

The terms of the pure lambda calculus are defined inductively as follows.

1. Each variable x is a term.
2. If M is a term, then $(\lambda x)M$ is a term.
3. If M, N are terms, then MN is a term.

Note that functional application is written as concatenation, so that $fx = f(x)$. After all, nothing but habit prevents you from thinking of 2 as a function that maps x^2 to 4 and x^3 to 8.

Now I can get to the (fixed) point.

Proposition 8.1 (Lambda fixed-point theorem) $(\forall F)(\exists X)[FX = X]$.

The proof is elegance itself, compared with Kleene's fixed-point construction for the numbering of the partial recursive functions. Let F be given. Now define the term:

$$X = (\lambda x)[Fxx](\lambda x)[Fxx].$$

Now calculate:

$$\begin{aligned} X &= (\lambda x)[Fxx](\lambda x)[Fxx] \\ &= F((\lambda x)[Fxx](\lambda x)[Fxx]) \\ &= FX. \end{aligned}$$

This is one of the most familiar facts about the lambda calculus. You can now imagine Kleene trying to massage this familiar syntactic manipulation into the more complicated context in which functions operating on functions are traded for partial recursive functions that operate on indices of other functions.

Exercise 8.3 *Clearly, $MM = MM$, since each object is identical to itself. But MM also has a value. For example $(\lambda x)[x](\lambda x)[x] = (\lambda x)[x]$. Find an elegant example of an M such that the* value *of MM is also MM. Hint: think of the operation of applying a term to itself. If you apply this operation to itself, it applies itself to itself.*

This sort of zany stuff made people wonder if the lambda calculus is actually about *anything at all, or is just clever symbol-shifting. After all, how can self-application be an operation applied to itself? The question remained open until our own Dana Scott found a systematic semantics for the lambda calculus.*

8.2 Kleene Fun

We usually use the recursion theorem in tandem with the universal and *s-m-n* theorems. The recursion theorem can generate curiosities, like a partial recursive function that returns its own index on every input.

self-printing program. At first it seems easy to make a self-printing program: something like

$$print(progam).$$

But that won't do because what is printed is *program*, not the actual program *print(program)*. Now we start to wonder if it is possible. It looks like there might be an infinite referential regress, in which the program tries forever to refer to itself but always misses the outermost "print" command in its own program. We would like to say

$$print(me),$$

but most programming languages don't have self-referential personal pronouns like "me". Nonetheless, it is always possible to get the same, self-referential effect, as long as your programming language (i.e., numbering of *Part*) satisfies the $s-m-n$ and universal theorems. The projection function p_2 is partial recursive. So let

$$\phi_n = p_2^2.$$

Now apply the *s-m-n* theorem to obtain a total recursive s such that for all x,

$$\begin{aligned} \phi_{s(n,x)}(y) &\simeq \phi_n(x,y) \\ &\simeq p_2^2(x,y). \end{aligned}$$

Now let

$$g(x) = s(c_n(x), x) = s(n, x),$$

so g is total recursive. By the Kleene recursion theorem, we obtain an m such that

$$\phi_{g(m)} = \phi_m.$$

Thus, for each x:

$$\begin{aligned} \phi_m(x) &\simeq \phi_{g(m)}(x) \\ &\simeq \phi_{s(n,m)}(x) \\ &\simeq \phi_n(m,y) \\ &\simeq p_2^2(m,y) \\ &= m. \end{aligned}$$

Exercise 8.4 *Show that each partial recursive function ϕ_i has a finite variant ϕ_j that is "self-referential" in the sense that $\mu z.\phi_j \neq 0 = j$ and $\forall k > j, \phi_j(k) \simeq \phi_i(k)$. .*

Exercise 8.5 *Show that double recursion over the partial recursive functions yields a partial recursive function. Before we only said that it is "intuitively effective". By the Church-Turing thesis it follows that double-recursion is partial recursive. But we can now prove this fact formally, bypassing the Church-Turing thesis. Hint: follow the pattern of the preceding example. Write an expression for the recursion in which the recursive call is just a free variable. Apply* s-m-n *to this variable position and then apply Kleene's fixed point theorem. This is why it's called the "recursion theorem".*

The Kleene recursion theorem has far wider significance than these examples suggest. As we will see later, it is a powerful tool for locating the problem of induction inside of purely formal problems. As such, it allows for simple, intuitive proofs of facts that require convoluted, reductio arguments without it.