thy.1 The Fixed-Point Theorem

cmp:thy:fix:sec

Let's consider the halting problem again. As temporary notation, let us write $\ulcorner\varphi_x(y)\urcorner$ for $\langle x, y\rangle$; think of this as representing a "name" for the value $\varphi_x(y)$. With this notation, we can reword one of our proofs that the halting problem is undecidable.

Question: is there a computable function h, with the following property? For every x and y,

$$h(\ulcorner\varphi_x(y)\urcorner) = \begin{cases} 1 & \text{if } \varphi_x(y)\downarrow \\ 0 & \text{otherwise.} \end{cases}$$

Answer: No; otherwise, the partial function

$$g(x) \simeq \begin{cases} 0 & \text{if } h(\ulcorner\varphi_x(x)\urcorner) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

would be computable, and so have some index e. But then we have

$$\varphi_e(e) \simeq \begin{cases} 0 & \text{if } h(\ulcorner\varphi_e(e)\urcorner) = 0 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

in which case $\varphi_e(e)$ is defined if and only if it isn't, a contradiction.

Now, take a look at the equation with φ_e. There is an instance of self-reference there, in a sense: we have arranged for the value of $\varphi_e(e)$ to depend on $\ulcorner\varphi_e(e)\urcorner$, in a certain way. The fixed-point theorem says that we *can* do this, in general—not just for the sake of proving contradictions.

Lemma thy.1 gives two equivalent ways of stating the fixed-point theorem. Logically speaking, the fact that the statements are equivalent follows from the fact that they are both true; but what we really mean is that each one follows straightforwardly from the other, so that they can be taken as alternative statements of the same theorem.

cmp:thy:fix:lem:fixed-equiv

Lemma thy.1. *The following statements are equivalent:*

1. *For every partial computable function $g(x, y)$, there is an index e such that for every y,*
$$\varphi_e(y) \simeq g(e, y).$$

2. *For every computable function $f(x)$, there is an index e such that for every y,*
$$\varphi_e(y) \simeq \varphi_{f(e)}(y).$$

Proof. (1) ⇒ (2): Given f, define g by $g(x, y) \simeq \mathrm{Un}(f(x), y)$. Use (1) to get an index e such that for every y,

$$\begin{aligned} \varphi_e(y) &= \mathrm{Un}(f(e), y) \\ &= \varphi_{f(e)}(y). \end{aligned}$$

(2) ⇒ (1): Given g, use the *s-m-n* theorem to get f such that for every x and y, $\varphi_{f(x)}(y) \simeq g(x, y)$. Use (2) to get an index e such that

$$\begin{aligned}\varphi_e(y) &= \varphi_{f(e)}(y)\\ &= g(e, y).\end{aligned}$$

This concludes the proof. □

explanation

Before showing that statement (1) is true (and hence (2) as well), consider how bizarre it is. Think of e as being a computer program; statement (1) says that given any partial computable $g(x, y)$, you can find a computer program e that computes $g_e(y) \simeq g(e, y)$. In other words, you can find a computer program that computes a function that references the program itself.

Theorem thy.2. *The two statements in Lemma thy.1 are true. Specifically, for every partial computable function $g(x, y)$, there is an index e such that for every y,*

$$\varphi_e(y) \simeq g(e, y).$$

Proof. The ingredients are already implicit in the discussion of the halting problem above. Let $\mathrm{diag}(x)$ be a computable function which for each x returns an index for the function $f_x(y) \simeq \varphi_x(x, y)$, i.e.

$$\varphi_{\mathrm{diag}(x)}(y) \simeq \varphi_x(x, y).$$

Think of diag as a function that transforms a program for a 2-ary function into a program for a 1-ary function, obtained by fixing the original program as its first argument. The function diag can be defined formally as follows: first define s by

$$s(x, y) \simeq \mathrm{Un}^2(x, x, y),$$

where Un^2 is a 3-ary function that is universal for partial computable 2-ary functions. Then, by the *s-m-n* theorem, we can find a primitive recursive function diag satisfying

$$\varphi_{\mathrm{diag}(x)}(y) \simeq s(x, y).$$

Now, define the function l by

$$l(x, y) \simeq g(\mathrm{diag}(x), y).$$

and let $\ulcorner l \urcorner$ be an index for l. Finally, let $e = \mathrm{diag}(\ulcorner l \urcorner)$. Then for every y, we have

$$\begin{aligned}\varphi_e(y) &\simeq \varphi_{\mathrm{diag}(\ulcorner l \urcorner)}(y)\\ &\simeq \varphi_{\ulcorner l \urcorner}(\ulcorner l \urcorner, y)\\ &\simeq l(\ulcorner l \urcorner, y)\\ &\simeq g(\mathrm{diag}(\ulcorner l \urcorner), y)\\ &\simeq g(e, y),\end{aligned}$$

as required. □

What's going on? Suppose you are given the task of writing a computer program that prints itself out. Suppose further, however, that you are working with a programming language with a rich and bizarre library of string functions. In particular, suppose your programming language has a function diag which works as follows: given an input string s, diag locates each instance of the symbol 'x' occuring in s, and replaces it by a quoted version of the original string. For example, given the string explanation

```
hello x world
```

as input, the function returns

```
hello 'hello x world' world
```

as output. In that case, it is easy to write the desired program; you can check that

```
print(diag('print(diag(x))'))
```

does the trick. For more common programming languages like C++ and Java, the same idea (with a more involved implementation) still works.

We are only a couple of steps away from the proof of the fixed-point theorem. Suppose a variant of the print function print(x, y) accepts a string x and another numeric argument y, and prints the string x repeatedly, y times. Then the "program"

```
getinput(y); print(diag('getinput(y); print(diag(x), y)'), y)
```

prints itself out y times, on input y. Replacing the getinput—print—diag skeleton by an arbitrary funtion $g(x, y)$ yields

```
g(diag('g(diag(x), y)'), y)
```

which is a program that, on input y, runs g on the program itself and y. Thinking of "quoting" with "using an index for," we have the proof above.

For now, it is o.k. if you want to think of the proof as formal trickery, or black magic. But you should be able to reconstruct the details of the argument given above. When we prove the incompleteness theorems (and the related "fixed-point theorem") we will discuss other ways of understanding why it works.

The same idea can be used to get a "fixed point" combinator. Suppose you have a lambda term g, and you want another term k with the property that k is β-equivalent to gk. Define terms digression

$$\text{diag}(x) = xx$$

and

$$l(x) = g(\text{diag}(x))$$

using our notational conventions; in other words, l is the term $\lambda x.\, g(xx)$. Let k be the term ll. Then we have

$$\begin{aligned} k &= (\lambda x.\, g(xx))(\lambda x.\, g(xx)) \\ &\triangleright\, g((\lambda x.\, g(xx))(\lambda x.\, g(xx))) \\ &= gk. \end{aligned}$$

If one takes

$$Y = \lambda g.\, ((\lambda x.\, g(xx))(\lambda x.\, g(xx)))$$

then Yg and $g(Yg)$ reduce to a common term; so $Yg \equiv_\beta g(Yg)$. This is known as "Curry's combinator." If instead one takes

$$Y = (\lambda xg.\, g(xxg))(\lambda xg.\, g(xxg))$$

then in fact Yg reduces to $g(Yg)$, which is a stronger statement. This latter version of Y is known as "Turing's combinator."

Photo Credits

Bibliography