14 The Least Search Operator

What happens if we eliminate the bound in bounded minimization? Would that be a computable operation?

A. The μ-Operator

Eliminating the bound leads to a problem of undefined points. Consider

$f(x)$ = the least y such that $y + x = 10$

For each $x > 10$, $f(x)$ is undefined. Yet f is still computable: for $f(12)$, for instance, we can check each y in turn to see that $y + 12 \neq 10$.

You might say that it's obvious that there is no such y that makes $f(12)$ defined. So why can't we use that fact to make a better function which is defined everywhere? That would be stepping outside the system. We'd need not only a program, an instruction for f that would tell us to calculate $y + x$ and verify whether it equals 10 or not, but also a *proof* that there is no such y if $x > 10$. In this case that would be easy. But consider the function

$h(0) = h(1) = 0$, and for $w > 1$,
$h(w)$ = the least $\langle x, y\rangle$ such that x and y are prime and $x + y = 2 \cdot w$

At present no one knows for which w this function is defined (the claim that it is defined for all w is called "Goldbach's Conjecture"). Yet for, say, $w = 4318$ we can constructively check in turn each pair $\langle x, y\rangle$ to see whether x and y are prime and $x + y = 2 \cdot 4318$.

We define the *least search operator*, also called the *μ-operator*, as:

$$\mu y\,[\,f(\vec{x}, y) = 0\,] = z \quad \text{iff} \quad \begin{cases} f(\vec{x}, z) = 0 \text{ and} \\ \text{for every } y < z,\ f(\vec{x}, y) \text{ is defined and} > 0 \end{cases}$$

B. The min-Operator

A comparison: Denote by "$\min_y [\,f(\vec{x}, y) = 0\,]$" the smallest solution to the equation $f(\vec{x}, y) = 0$ if such exists, and undefined otherwise.

The following example shows that the min-operator is not the same as the μ-operator. Define the primitive recursive function:

$$h(x,y) = \begin{cases} x - y & \text{if } y \le x \\ 1 & \text{otherwise} \end{cases}$$

Now define:

$$g(x) = \mu y [\, 2 \dot{-} h(x,y) = 0 \,]$$
$$g^*(x) = \min_y [\, 2 \dot{-} h(x,y) = 0 \,]$$

Then

$g(0), g(1)$ are undefined, $g(2) = 0$

$g^*(0), g^*(1)$ are undefined, $g^*(2) = 0$

But now define:

$$f(x) = \mu y [\, g(y) \cdot (x+1) = 0 \,]$$
$$f^*(x) = \min_y [\, g^*(y) \cdot (x+1) = 0 \,]$$

Then $f(x)$ is undefined for all x; but for all x, $f^*(x) = 2$.

C. The μ-Operator Is a Computable Operation

Why do we choose the μ-operator rather than the min-operator? We might not be able to predict for which x, $f(x,y) = 0$ has a solution. To say that $\min_y [\, f(x,y) = 0 \,] = 1$ when $f(x,0)$ is undefined due to an infinite search entails the completion of an infinite task.

But with the μ-operator if $f(x,0)$ is undefined (that is, we're put into a search that never ends) then $\mu y [\, f(x,y) = 0 \,]$ is undefined too, for we never get a shot at trying $f(x,1) = 0$.

```
To calculate $g(x) = \mu y [\, f(x,y) = 0 \,]$ we proceed in steps:
Step 0: calculate $f(x,0)$
        if defined $= 0$ output the number of this step;
        if defined $> 0$ we continue the search at the next step;
        if we never get an answer for $f(x,0)$ then $g(x)$ will be undefined,
        for it is in an unending search
Step 1: calculate $f(x,1)$ —proceed as in step 0
Step 2: calculate $f(x,2)$ —proceed as in step 0
   ⋮
```

This is a well-defined *computable procedure* for calculating $g(x)$, though it may not always give a result.

15 Partial Recursive Functions

A. The Partial Recursive Functions

1. Our investigations have led us to make the following definition:

> The *partial recursive functions* are the smallest class containing the zero, successor, and projection functions and closed under composition, primitive recursion, and the μ-operator.

Again, we remind you that even though phrased in terms of classes, this is really an inductive definition of the label "partial recursive" (see Chapter 11.B). We sometimes abbreviate "partial recursive" as *p. r.*

Since our functions may not be defined on all inputs, we will say that a function that is defined for all inputs is *total* and will continue to use lowercase Roman letters, *f*, *g*, *h*, etc. for total functions. Note that any total function we have so far investigated is partial recursive: We can define it just as before simply deleting any reference to a bound. The term *recursive* is reserved for total p. r. functions.

We call functions which may (for all we know) be undefined for some inputs *partial* functions and use lowercase Greek letters φ , ψ , ρ and so on, to denote them. We write, for example, $\varphi(x)$ to mean the function (thought of as a procedure) applied to x. We do not necessarily mean by this that there is an object called $\varphi(x)$, for φ applied to x may be undefined. We write

$\varphi(x)\downarrow$ for "φ applied to x is defined"
$\varphi(x)\not\downarrow$ for "φ applied to x is not defined"

When are two partial functions (extensionally) the same? First, φ and ψ *agree on input* x if both $\varphi(x)\downarrow$ and $\psi(x)\downarrow$ and these are equal, or both $\varphi(x)$ and $\psi(x)$ are undefined. In that case, we write:

$$\varphi(x) \simeq \psi(x)$$

We say that φ and ψ are the same function if they agree on all inputs; that is, for all x, $\varphi(x) \simeq \psi(x)$. In that case, we write:

$$\varphi \simeq \psi$$

These conventions also apply to functions of several variables.

We say that a *set* A or *relation* R *is recursive* if its characteristic function, C_A or C_R, is recursive (see Chapter 11.D.2 and D.3). Note that every characteristic function is total (that's the law of excluded middle).

When we use the μ-operator we need to reverse the roles of 0 and 1 in the characteristic function, so we define the *representing function* for a relation R to be $\overline{sg} \circ C_R$.

2. It is not as restrictive as it may appear that the μ-operator requires us to search for a y such that $\varphi(\vec{x}, y) = 0$. Given a relation R, we write:

$$\mu y_{\le g(\vec{x})}\, [R(\vec{x}, y)] \quad \text{to mean} \quad \mu y\, [y \le g(\vec{x}) \wedge R(\vec{x}, y)]\,.$$

Lemma 1 If g and R are recursive, then the following functions are partial recursive.

$$\begin{aligned}
\tau(\vec{x}) &\simeq \mu y\, [\, \varphi(\vec{x}, y) = a\,] \\
\rho(\vec{x}) &\simeq \mu y\, [R(\vec{x}, y)] \\
\psi(\vec{x}) &\simeq \mu y_{\le g(\vec{x})}\, [R(\vec{x}, y)] \\
\gamma(\vec{x}) &\simeq \mu y_{< g(\vec{x})}\, [R(\vec{x}, y)]
\end{aligned}$$

We leave the proof as Exercise 1.

B. Diagonalization and the Halting Problem

Why can't we diagonalize out of this class?

Assume for the moment that we can effectively number all the partial recursive functions of one variable as $\varphi_1, \varphi_2, \dots, \varphi_n, \dots$ (we'll indicate how in the next chapter, but from our previous experience with numbering this should be plausible). We can then define $\psi(x) = \varphi_x(x) + 1$. *But it won't diagonalize,* because some $\varphi_x(x)$ are *not defined.*

Can we avoid this and hence diagonalize by deciding if $\varphi_x(x)$ is defined?

THEOREM 2 There is no recursive function which can tell us whether $\varphi_x(x)$ is defined.

Proof: Suppose such a recursive function, f, exists:

$$f(x) \simeq \begin{cases} 1 & \text{if } \varphi_x(x)\downarrow \\ 0 & \text{if } \varphi_x(x)\not\downarrow \end{cases}$$

Then

$$\rho(x) \simeq \begin{cases} \not\downarrow & \text{if } \varphi_x(x)\downarrow \\ 0 & \text{if } \varphi_x(x)\not\downarrow \end{cases}$$

is partial recursive. For the first and last time we will formally define such a function: $\rho(x) \simeq \mu y\, [\, y + f(x) = 0\,]$. So ρ must be φ_y for some y. But

$$\rho(y) = \begin{cases} \uparrow & \text{if } \varphi_y(y)\downarrow \\ 0 & \text{if } \varphi_y(y)\uparrow \end{cases}$$

This is a contradiction. So no such f exists. ■

Diagonalization bites the dust—but at the cost of introducing partial functions!

We've shown that $K \equiv_{\text{Def}} \{ x : \varphi_x(x)\downarrow \}$ is not recursive. More generally, define $K_0 \equiv_{\text{Def}} \{ \langle x, y \rangle : \varphi_x(y)\downarrow \}$. The characteristic function of K_0 is called the *halting problem* for the partial recursive functions since $\langle x, y \rangle$ is in K_0 iff the x^{th} algorithm applied to y halts. We leave the proof of the following as Exercise 4.

COROLLARY 3 (The Halting Problem Is Unsolvable) K_0 is not recursive.

C. The General Recursive Functions

Remember when you were asked to say what you thought the criteria should be for a procedure to be computable (Chapter 8)? If you said it should terminate, you must be very dissatisfied with our introduction of partial functions.

Let's say a function $g(\vec{x}, y)$ is *regular* if it is total and for every $\vec{x}$ there is some y such that $g(\vec{x}, y) = 0$. The class of *general recursive functions* is defined exactly as the partial recursive ones except that the μ-operator may be applied only to regular functions. Clearly, every general recursive function is a total function, indeed, a total partial recursive function.

Church's Thesis (in one of its equivalent forms) asserts:
A function is computable iff it is general recursive.

But it is a false pleasure we get from creating a class all of whose functions are total. The operation of μ-operator applied to regular functions is "ill-defined" because for arbitrary x *we cannot decide* if $g(x, y) = 0$ has a solution. That might require calculating each of $g(x,0)$, $g(x,1)$, $g(x,2)$, ... none of which may equal 0 (and, note, it would entail solving the halting problem). So we *cannot effectively number* the general recursive functions (and hence can't computably diagonalize them either). The ambiguity of partial functions is essential in order to obtain a class of computable functions we can computably number and yet not computably diagonalize. In Chapter 16 we'll show:

the general recursive functions $\equiv$ *the total partial recursive functions*

D. Gödel on Partial Functions

Gödel points out that the precise notion of mechanical procedures is brought out clearly by Turing machines producing partial rather than general recursive functions. In other words, the intuitive notion does not require that a mechanical procedure should always terminate or succeed. A sometimes unsuccessful procedure, if sharply defined, still is a procedure, i.e. a well determined manner of proceeding. Hence we have an excellent example here of a concept which did not appear sharp to us but has become so as a result of a careful reflection. The resulting definition of the concept of mechanical by the sharp concept of 'performable by a Turing machine' is both correct and unique. Unlike the more complex concept of always-terminating mechanical procedures, the unqualified concept, seen clearly now, has the same meaning for the intuitionists [a brand of constructive mathematicians, see Chapter 26] as for the classicists. Moreover it is absolutely impossible that anybody who understands the question and knows Turing's definition should decide for a different concept.

Wang, p. 84

Exercises

1. Prove Lemma 1 (cf. Exercise 11.15).

2. Show that the function defined by

$$g(x)=\begin{cases}0 & \text{if } x \text{ is even}\\ \not\downarrow & \text{if } x \text{ is odd}\end{cases}$$

   is partial recursive by giving a μ-operator definition of it using functions which we've already shown are partial recursive.

3. a. Applied to regular functions the min-operator and the μ-operator are the same. Suppose that φ is partial recursive and f is defined by $f(\vec{x}) = \mu y[\varphi(\vec{x}, y)=0]$. If f is total, show that for all $\vec{x}$, $f(\vec{x}) = \min y\,[\varphi(\vec{x}, y)=0]$.

   b. Show that the partial recursive functions are not closed under the min-operator. (*Hint:* Define $\psi(x, y) = 1$ if either $y = 1$, or both $y = 0$ and $\varphi_x(x)\downarrow$.)

4. Prove that K_0 is not recursive.

5. We say that a function ψ *extends* a function φ if whenever $\varphi(x)\downarrow$, $\psi(x)\downarrow = \varphi(x)$. Show that there is a partial recursive function φ which cannot be extended to a total recursive function. (*Hint:* Consider $\varphi \simeq \lambda x(\,\varphi_x(x) + 1\,)$.)

6. We said that we can't diagonalize out of the class of partial recursive functions. But doesn't the function f defined by

$$f(x)=\begin{cases}\varphi_x(x)+1 & \text{if } \varphi_x(x)\downarrow\\ 0 & \text{otherwise}\end{cases}$$

   diagonalize the class of partial recursive functions? Explain.

7. Explain why you do or do not agree that partial functions are computable. Are Gödel's remarks (via Wang) a convincing argument for accepting partial procedures as computable ?

16 Numbering the Partial Recursive Functions

A. Why and How: The Idea

We want to number the partial recursive functions for two reasons. First, we want to justify and make precise the comments we made in Chapter 15 about the halting problem and diagonalization. Second, as we number the functions we will code how each is built up, so that given the number of a function we can uncode it to compute the function on any input. In this way we will have a partial recursive procedure which simulates all partial recursive functions, what we call a "universal function" for the partial recursive functions.

The idea of the numbering isn't hard; it's no harder than the sketch we made of numbering the primitive recursive functions in Chapter 11.F. But writing it down gets a bit complicated, so we'll give the idea in rough form first. We're going to use the coding of sequences of numbers we presented in Chapter 11.E.5 with which you should be familiar.

The numbering is an inductive procedure. As basis, we number the initial functions, say Z gets number 0, S gets number $\langle 1 \rangle$, and $P_k^{\,i}$ gets number $\langle 1, i \rangle$ (the number of variables will determine k). At the induction stage we assume that we have already numbered various functions, say φ_a, φ_b, φ_{b_1}, $\varphi_{b_2}, \ldots, \varphi_{b_k}$. Then corresponding to each operation we can use to produce new functions, we will associate an arithmetical operation:

for composition, $\varphi_a(\varphi_{b_1}, \varphi_{b_2}, \ldots, \varphi_{b_k})$ will get number $\langle a, \langle b_1, b_2, \ldots, b_k \rangle, 0 \rangle$

for primitive recursion, the function defined by recursion on φ_b with basis φ_a will get number $\langle a, b, 0, 0 \rangle$

for the least search operator, $\mu y\,(\varphi_a(\vec{x}, y) = 0)$ will get number $\langle a, 0, 0, 0, 0 \rangle$

There are two complications, however, that make the numbering harder than this sketch. First, we will want every number to be the number of some function, so we will build in a lot of redundancy. For example, rather than assigning just $\langle a,0,0,0,0\rangle$ to the function that arises by an application of the μ-operator to φ_a, we'll assign every n with $lh(n) \geq 5$ such that $(n)_0 = a$. Second, we need to number all functions of every possible number of variables at one go. Thus, when we come to the number n we will have to stipulate for every k what the n^{th} function of k variables is. This is analogous to defining a Turing machine that works for inputs of any number of variables.

Now, given any number we can unpack it to see what function it corresponds to. For example, if we have $n = \langle a,0,0,0,0\rangle$ then we know that it's the number of a function which arises by least search operator applied to φ_a, where $a < n$. We can continue to unpack, say $a = \langle 4796521, 814, 0, 0\rangle$. In this case, we know that φ_a is obtained by primitive recursion on φ_{814} with basis $\varphi_{4796521}$. By unpacking n until we arrive at a complete description of it using only primes, 0, and 1, we can get a description of the function it indexes in terms of initial functions and operations on them. Hence, given any x we can describe how to calculate $\varphi_n(x)$. The formal description of that process amounts to a universal function for the partial recursive functions.

B. Indices for the Partial Recursive Functions

Recall that $\varphi \simeq \psi$ means that for all x, $\varphi(x)\downarrow$ iff $\psi(x)\downarrow$, and they are equal if both are defined.

We shall index all partial recursive functions of every number of variables at once. The n^{th} function of k variables for $k \geq 1$ will be denoted φ_n^k, though we will drop the k if it's clear. The variables will always be labeled $x_1, x_2, \ldots, x_k$. We will give a definition by inducting on n for all k at once.

By induction, we define $\varphi_n^k(x_1, x_2, \ldots, x_k)$.

Case 1 (zero, successor) $lh(n) = 0$ or 1
n codes the empty sequence or $(n)_0$
If $lh(n) = 0$, then $\varphi_n^k \simeq$ the constant function Z
If $lh(n) = 1$, then $\varphi_n^k \simeq$ the successor function, $x_1 + 1$

Case 2 (projections) $lh(n) = 2$
n codes the sequence $((n)_0, (n)_1)$
If $(n)_0 = 0$ or $(n)_1 = 0$, then $\varphi_n^k \simeq$ the constant function Z
If $(n)_0 \geq 1$ and $1 \leq (n)_1 \leq k$, then $\varphi_n^k \simeq P_k^{(n)_1}(x_1, x_2, \ldots, x_k)$
If $(n)_0 \geq 1$ and $k < (n)_1$, then $\varphi_n^k \simeq P_k^{k}(x_1, x_2, \ldots, x_k)$

Case 3 (composition of functions) $lh(n) = 3$
n codes the sequence $((n)_0, (n)_1, (n)_2)$
If $lh((n)_1) = 0$, then $\varphi_n^k \simeq \varphi_{(n)_0}^k$
If $lh((n)_1) \geq 1$, then $\varphi_n^k \simeq \varphi_{(n)_0}^{lh((n)_1)}(\varphi^k_{((n)_1)_0}, \ldots, \varphi^k_{((n)_1)_{lh((n)_1) \dot{-} 1}})$

Case 4 (primitive recursion) $lh(n) = 4$

n codes the sequence $((n)_0, (n)_1, (n)_2, (n)_3)$

If $k = 1$, then

$$\varphi_n(0) \simeq (n)_3$$
$$\varphi_n(x_1 + 1) \simeq \varphi^2_{(n)_1}(\varphi_n(x_1), x_1)$$

If $k > 1$, then

$$\varphi^k_n(0, x_2, \ldots, x_k) \simeq \varphi^{k-1}_{(n)_0}(x_2, \ldots, x_k)$$
$$\varphi^k_n(x_1 + 1, x_2, \ldots, x_k) \simeq$$
$$\varphi^{k+1}_{(n)_1}(\varphi^k_n(x_1, x_2, \ldots, x_k), x_1, x_2, \ldots, x_k)$$

Case 5 (μ-operator) $lh(n) \geq 5$

n codes the sequence $((n)_0, \ldots, (n)_{lh(n)_1 \dot{-} 1})$

$$\varphi^k_n(x_1, x_2, \ldots, x_k) \simeq \mu x_{k+1}[\varphi^{k+1}_{(n)_0}(x_1, x_2, \ldots, x_k, x_{k+1}) = 0]$$

This completes the numbering. If $\psi \simeq \varphi^k_n$, we call n an *index* of ψ.

Note that we have numbered programs, descriptions of functions. In Exercise 2 you're asked to prove that every partial recursive function has arbitrarily many different indices. This is not an accidental feature of the numbering: In Exercise 9 you're asked to show that there is no recursive way to determine whether two programs give the same function.

THEOREM 1 φ is partial recursive iff for some n, $\varphi \simeq \varphi_n$.

Proof: We leave to you to show by induction on n that for every n, φ_n is partial recursive.

To show that if φ is partial recursive there is some n such that $\varphi \simeq \varphi_n$, we use *induction on the number of applications of the basic operations in a definition of* φ.

If no operations are used in the definition then φ is an initial function: if φ is Z then $\varphi \simeq \varphi_0$; if φ is S then $\varphi \simeq \varphi_4$; and if φ is P^i_k then $\varphi \simeq \varphi_{4 \cdot 3^i + 1}$.

Suppose now that the theorem is true for every function which has a definition using at most m applications of the basic operations and φ has a definition which uses $m + 1$ applications. If φ is defined by an application of the μ-operator applied to ρ, and ρ has a definition with at most m applications of the basic operations, then for some r, $\rho \simeq \varphi_r$ and $\varphi \simeq \varphi_{2^{r+1} \cdot 3 \cdot 5 \cdot 7 \cdot 11}$. The other cases are similar and we leave them to you as Exercise 1. ∎

C. Algorithmic Classes (Optional)

Generally, we call any class of functions of natural numbers *algorithmic* if it contains the zero, successor, and projection functions, and is closed under the operations of composition, primitive recursion, and μ-operator.

The class of partial recursive functions is the smallest algorithmic class. In set theoretic terminology, it is the intersection of all algorithmic classes, which is to say that every partial recursive function is contained in every algorithmic class. Thus, the difference between a general algorithmic class and the partial recursive functions is what additional nonrecursive initial functions are chosen.

If $f_1, f_2, \ldots, f_n$ are total functions of the natural numbers, then we say that φ is *partial recursive in* $\{f_1, f_2, \ldots, f_n\}$ iff φ can be obtained from the initial functions zero and successor, the projections, and $f_1, f_2, \ldots, f_n$ by the operations of composition, primitive recursion, and μ-operator.

For Exercise 11 we ask you to give a numbering of the functions partial recursive in $\{f_1, f_2, \ldots, f_n\}$.

D. The Universal Computation Predicate

Using our numbering, we'd like to check whether $\varphi_n(b)\downarrow = r$. But we can't because the halting problem is unsolvable (Corollary 15.3). However, *if* $\varphi_n(b)\downarrow = r$, we can tell that: since n codes the definition of the function, we can actually do the computation. But $\varphi_n(b)$ might be undefined due to an infinite search. We will get a recursive predicate by limiting the searches we can do to check whether $\varphi_n(b)\downarrow = r$. So "$C(n, b, r, q)$" will mean that $\varphi_n(b)\downarrow = r$ and q bounds the largest number used in that computation. Thus if $C(n, b, r, q)$ holds, so will $C(n, b, r, w)$ for any $w > q$ (intuitively, if you can compute in time q and $q < w$, then you can compute in time w). Moreover, because all the searches are bounded by q, the predicate that checks the computation will actually be primitive recursive. This is an important point: there is no effective procedure for determining whether $\varphi_n(b)\downarrow = r$, but given a purported computation, "$\varphi_n(b)\downarrow = r$ in time q", we can check it primitive recursively. Indeed, the checking process is elementary (Corollary 3).

When we write "$C(n, b, r, q)$" we mean to assert that $C(n,b,r,q)$ holds.

THEOREM 2 (The Universal Computation Predicate)
There is a primitive recursive predicate C such that

$$\varphi_n^k(b_1, \ldots, b_k) = r \text{ iff } \exists q\, C(n, \langle b_1, \ldots, b_k \rangle, r, q)$$

Moreover, if $C(n, b, r, q)$ and $q < w$, then $C(n, b, r, w)$.

Proof: We are going to define C by induction on n (the number of the function) by stipulating those cases in which it holds (and thus by implication in all others it fails). Thus, when we define $C(n,b,r,q)$ we may assume that we have already defined $C(m,x,y,z)$ for any $m < n$ and all x, y, z. We can do this because the only functions which are referred to in the definition of φ_n are φ_m for $m < n$. Also note that if $C(n,b,r,v)$ and $lh(b) = k \geq 1$, then b will code $(b_1, \ldots, b_k)$ so that k is the number of variables involved (the variables are numbered starting with 1, the primes starting with $p_0 = 2$).

Case 1 $lh(n) = 0$ and $r = 0$
or $lh(n) = 1$ and $r = (b)_0 + 1$

Case 2 $lh(n) = 2$ and
$(n)_0 = 0$ or $(n)_1 = 0$, and $r = 0$
or $(n)_0 \geq 1$ and $1 \leq (n)_1 \leq lh(b)$ and $r = (b)_{(n)_1 \dot{-} 1}$
or $(n)_0 \geq 1$ and $\mathrm{k} < (n)_1$ and $r = (b)_{lh(b) \dot{-} 1}$

Note that q is irrelevant in cases 1 and 2 .

Case 3 $lh(n) = 3$ and
$lh((n)_1) = 0$ and $C((n)_0, b, r, q)$
or $lh((n)_1) \geq 1$ and
$\exists d \leq q$ with $lh(d) = lh((n)_1)$ and
$C(((n)_1)_i, b, (d)_i, q)$ for $0 \leq i \leq lh((n)_1) \dot{-} 1$
and $C((n)_0, d, r, q)$

Case 4 $lh(n) = 4$ and
$lh(b) = 1$ and $(b)_0 = 0$ and $r = (n)_3$
or $lh(b) = 1$ and $(b)_0 \geq 1$ and $\exists e \leq q$,
$C(n, \frac{b}{2}, e, q)$ and $C((n)_1, \langle e, (b)_0 \dot{-} 1 \rangle, r, q)$
or $lh(b) > 1$ and $(b)_0 = 0$ and
$C((n)_0, \langle (b)_1, \ldots, (b)_{lh(b) \dot{-} 1} \rangle, r, q)$
or $lh(b) > 1$ and $(b)_0 \geq 1$ and $\exists e \leq q$,
such that $C(n, \frac{b}{2}, e, q)$ and
$C((n)_1, \langle e, (b)_0 \dot{-} 1, (b)_1, \ldots, (b)_{lh(b) \dot{-} 1} \rangle, r, q)$

Case 5 $lh(n) \geq 5$ and
$C((n)_0, b \cdot p(lh(b))^{1+r}, 0, q)$ [recall that $p(x) =$ the x^{th} prime]
and $\forall\, i < r$, $\exists e$, $0 < e < q$, such that
$C((n)_0, b \cdot p(lh(b))^{1+i}, e, q)$

This completes the description of C.

Now we must prove that C does what we claim. First, note that C is primitive recursive since every condition is obtained by bounded existence on some primitive recursive condition.

To show that $\varphi_n(b_1, \ldots, b_k) = r$ iff $\exists q\, C(n, \langle b_1, \ldots, b_k \rangle, r, q)$ we induct on n and subinduct on $b = \langle b_1, \ldots, b_k \rangle$. For the basis, we note that for $lh(n) \leq 2$ it's clear.

Suppose now it is true for all $a < n$ and for n for all $x < b$. We'll do only one direction of one case, and leave the rest to you. Suppose $\varphi_n(b_1, \ldots, b_k) = r$

and $lh(n) = 5$. Then $\varphi_{(n)_0}(b_1, \ldots, b_k, r) = 0$ and hence, since $(n)_0 < n$, by induction there is some v such that $C((n)_0, \langle b_1, \ldots, b_k, r\rangle, 0, v)$. Moreover, for each $i < r$, $\varphi_{(n)_0}(b_1, \ldots, b_k, i)\downarrow > 0$. So by induction we know that there are $u_0, \ldots, u_{r-1}$ and $v_0, \ldots, v_{r-1}$ such that for all $i \le r-1$, $u_i > 0$ and $C((n)_0, \langle b_1, \ldots, b_k, i\rangle, u_i, v_i)$. Take

$$q = max(u_0, \ldots, u_{r-1}, v_0, \ldots, v_{r-1}, v, r) + 1$$

Then $C(n, \langle b_1, \ldots, b_k\rangle, r, q)$. ■

We only claimed in Theorem 2 that C is primitive recursive, but actually we've proved more. Recall that a function is elementary if it is in $\mathcal{E}$ (Chapter 12.B).

COROLLARY 3 The universal computation predicate is elementary.

Proof: This is just a matter of tracing through the definition of C to see that every condition is obtained by bounded existence on some elementary condition. ■

Since the representing function (total) for the universal computation predicate is partial recursive it must have an index. We call the least one c, so that

$$\varphi_c^4(n, m, r, q) = \begin{cases} 0 & \text{if } C(n, m, r, q) \\ 1 & \text{if not } C(n, m, r, q) \end{cases}$$

E. The Normal Form Theorem

What we have done so far in this chapter may seem merely a tedious exercise in labeling and reading labels. But the names we have given code a lot of information. Using the numbering we can define a universal partial recursive function, one which calculates all others, analogous to one Turing machine which simulates all others.

THEOREM 4 For $\vec{x} = (x_1, x_2, \ldots, x_k)$ the function

$$\lambda n, \vec{x} \; (\mu q [C(n, \langle \vec{x}\rangle, (q)_0, q)])_0$$

is partial recursive and is universal for the partial recursive functions of k variables. That is, if φ is a partial recursive function of k variables, then for some n, all $\vec{x}$,

$$\begin{aligned} \varphi(\vec{x}) &\simeq (\mu q [C(n, \langle \vec{x}\rangle, (q)_0, q)])_0 \\ &\simeq (\mu q [\varphi_c^4(n, \langle \vec{x}\rangle, (q)_0, q) = 0])_0 \end{aligned}$$

Proof: By Theorem 1, if φ is partial recursive then for some n, φ is φ_n. So by Theorem 2, if $\varphi(\vec{x})\downarrow = r$, then there is some s such that $C(n, \langle \vec{x}\rangle, r, s)$. Hence $C(n, \langle \vec{x}\rangle, r, \langle r, s\rangle)$ and the theorem follows. ■

We know that every general recursive function is partial recursive, but in Chapter 15 we said that we could prove the converse. Using the Normal Form Theorem that's now easy.

COROLLARY 5 **a.** Every partial recursive function may be defined with at most one use of the μ-operator.
b. A total function is partial recursive iff it is general recursive.

Proof: a. This part follows from Theorem 4 since φ_c^4 is primitive recursive.

b. Given any total partial recursive function, by Theorem 4 there is a definition of it that uses only one application of the μ-operator applied to a primitive recursive function. Hence that primitive recursive function must be regular. ■

F. The *s-m-n* Theorem

Consider the partial recursive function

$$\varphi(x,y) \simeq x^y + [y \cdot \varphi_x^1(y)]$$

(Exercise 4). Suppose we take y as a parameter and consider, for example,

$$\lambda x(x^3 + [3 \cdot \varphi_x^1(3)])$$

Then that's partial recursive too. Generally, by using the Normal Form Theorem we can show that if we start with a partial recursive function and hold one or several variables fixed, we get another partial recursive function. More importantly, though, we can find an index for the new function effectively in the index of the given one.

THEOREM 6 (The *s-m-n* Theorem) For every $n, m \geq 1$ there is a recursive function S_n^m such that if we hold the first m variables fixed in

$$\varphi_x(a_1, \dots, a_m, y_1, \dots, y_n)$$

then an index for the resulting partial recursive function is $S_n^m(x, a_1, \dots, a_m)$.

$$\lambda y_1 \dots y_n [\varphi_x(a_1, \dots, a_m, y_1, \dots, y_n)]$$
$$\simeq \varphi_{S_n^m(x, a_1, \dots, a_m)}(y_1, \dots, y_n)$$

Proof: The left-hand side of the equation is

$$(\mu q [\varphi_c(x, \langle \underbrace{a_1, \dots, a_m}_{\text{view these as constants}}, \underbrace{y_1, \dots, y_n}_{\text{view these as projections}} \rangle, (q)_0, q) = 0])_0$$

To begin, the number 36 is an index for the identity function. In Exercise 5 we ask you to define a primitive recursive function h such that for all n, $\varphi_{h(n)}$ is the constant function $\lambda x(n)$. As before, $p(m) =$ the m^{th} prime. We'll let you calculate an index d for the function $\lambda x((x)_0)$. Using these we can define

$$S^m_n\,(x, a_1, \dots, a_m) = 2^{d+1} \cdot 3^{a+1} \cdot 5$$

where $a = 2^{b+1} \cdot 3 \cdot 5 \cdot 7 \cdot 11$ and $b = 2^{c+1} \cdot 3^{e+1} \cdot 5$ where

$$e = p(0)^{h(x)+1} \cdot p(1)^{h(a_1)+1} \cdot \dots \cdot p\,(m)^{h(a_m)+1} \cdot p(m+1)^{1+(2^2 \cdot 3^{1+1})} \cdot \dots \cdot p\,(m+n+1)^{1+(2^2 \cdot 3^{n+1})}$$

To see that this is correct requires going back to the numbering in Section B to check each part. That's a good way to get a grip on how the numbering works, and so we'll leave it to you.

Note that each S^m_n function is elementary since it involves only addition, multiplication, and composition on the functions p and h, all of which are elementary. ■

G. The Fixed Point Theorem

Self-reference can be a problem, as we know from the liar paradox and our many uses of diagonalization. But it has also been a useful tool for us, since that is exactly what primitive recursion is based on: A function is defined in terms of itself. And we have seen other forms of recursion in which a function could be defined in terms of itself in Chapters 11–13. Here we will show that the immunity to diagonalization that the partial recursive functions enjoy can be put to good use to find fixed points and hence very general ways to define a function in terms of itself.

THEOREM 7 (The Fixed Point Theorem)

If f is recursive then there is an e such that $\varphi_e \simeq \varphi_{f(e)}$.

Proof: Consider the function

$$\lambda xy \;\; \varphi_{\varphi_x(x)}(y) \simeq \begin{cases} \varphi_{\varphi_x(x)}(y) & \text{if } \varphi_x(x)\downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

This is partial recursive since it can be defined as

$$\psi(x, y) \simeq (\mu q\; C(\varphi_x(x), \langle y \rangle, (q)_0, q))_0$$

So by the *s-m-n* theorem there is a function g such that $\varphi_{g(x)} \simeq \lambda y\;\psi(x, y)$. Now consider $f \circ g$, which is recursive. For some d we have $f \circ g \simeq \varphi_d$. Hence by the definition of g,

$$\varphi_{g(d)} \simeq \varphi_{\varphi_d(d)}$$

and by the definition of d,

$$\varphi_{f(g(d))} \simeq \varphi_{\varphi_d(d)}$$

Thus we may take $e = g(d)$. ■

The Fixed Point Theorem is sometimes called the *Recursion Theorem.* Here is an example of its use, which we will need later. Although there are other ways to obtain the same result, this proof illustrates a typical application.

COROLLARY 8 If A is an infinite recursive set, then its elements can be enumerated in increasing order by a recursive function.

Proof: Let a be the least element of A. First note that the function we want is defined by the equations:

$$f(0) = a$$
$$f(x+1) = \mu y\,[y > f(x) \wedge y \in A]$$

Define a function of two variables ρ by:

$$\rho(i, 0) = a$$
$$\rho(i, x+1) \simeq \mu y\,[\, y > \varphi_i(x) \wedge y \in A]$$

We leave to you to show that ρ is partial recursive. The collection of functions $\lambda x\, \rho(i, x)$ gives us a matrix for which (instead of diagonalizing) we can find a fixed point. First, by the *s-m-n* theorem there is some recursive s such that $\varphi_{s(i)} \simeq \lambda x\, \rho(i, x)$. Then by the Fixed Point Theorem there is some e such that $\varphi_{s(e)} \simeq \varphi_e$. Hence

$$\varphi_e(0) = a$$
$$\varphi_e(x+1) \simeq \mu y\,[\, y > \varphi_e(x) \wedge y \in A] \qquad \blacksquare$$

Exercises

1. Complete the proof of Theorem 1.

2. Prove:
   a. There are exactly countably many different partial recursive functions.
   b. Every partial recursive function has arbitrarily many different indices.

3. We now have a universal partial recursive function. So from now on you don't have to write any more programs, just hand in an index. Right?

4. Show that the example we gave in Section F, $\varphi(x,y) \simeq x^y + [y \cdot \varphi^1_x(y)]$, is partial recursive. (*Hint:* Use the Normal Form Theorem.)

5. Define a primitive recursive function h such that for all n, $\varphi_{h(n)}$ is the constant function $\lambda x\,(n)$.

6. a. Give an index for addition.
   b. Give an index for multiplication.
   Express your answers in decimal notation.

†7. a. Show that there is a primitive recursive predicate *Prim* such that f is primitive recursive iff for some n, $f = \varphi^k_n$ and *Prim*(n).
   (*Note:* There may be some indices of addition, for example, which do not satisfy *Prim*, but at least one will.)
   b. Using part (a) show that there is a total recursive function which is not primitive recursive (cf. Chapter 11.G).

†8. The *graph* of a function φ is $\{ z : z = \langle x, \varphi(x) \rangle \}$. Show that there is a function with primitive recursive graph that is not primitive recursive.

9. Show that there is no recursive function which can identify whether two programs compute the same function. That is, show that no recursive function f satisfies

$$f(x, y) = \begin{cases} 1 & \text{if } \varphi_x \simeq \varphi_y \\ 0 & \text{if } \varphi_x \not\simeq \varphi_y \end{cases}$$

(*Hint:* Show that g such that

$$\varphi_{g(x)} \simeq \begin{cases} \lambda x\,(1) & \text{if } \varphi_x(x)\downarrow \\ \uparrow \text{ on all inputs} & \text{if } \varphi_x(x)\uparrow \end{cases}$$

is recursive, and look for a solution to the halting problem.)

10. Let $A = \{ x : \varphi_x^1 \text{ is total} \}$.
   a. Show that there is no recursive f such that A = range of f.
   (*Hint:* Diagonalize.)
   b. Show that A is not recursive. (*Hint:* Reduce this to part (a).)

11. Number the functions partial recursive in $\{ f_1, \ldots, f_n \}$. Using that, produce a universal computation predicate for the functions partial recursive in $\{ f_1, \ldots, f_n \}$ which is also partial recursive in $\{ f_1, \ldots, f_n \}$. What is the significance of this if $f_1, \ldots, f_n$ are recursive?

†12. Give an index for the universal partial recursive function of k variables defined in Theorem 4 in terms of the indices c, d, and e, where φ_c^4 is the representing function for the universal computation predicate, d is an index for $\lambda n\,(n)_0$, and e is an index for $\lambda \vec{x}\,\langle \vec{x} \rangle$.

13. We may delete primitive recursion as a basic operation in defining the partial recursive functions if we expand the class of initial functions. Show that the partial recursive functions are the smallest class containing the zero, successor, projection, and exponentiation functions and which is closed under composition and μ-operator.
(*Hint:* Use Corollary 3, Theorem 4, and Theorem 12.1.)

†14. *Rice's Theorem*
Prove: If C is a collection of partial recursive functions, then $\{ x : \varphi_x \text{ is in } C \}$ is a recursive set iff $C = \varnothing$ or C = all p. r. functions.
(*Hint:* Assume the contrary and $\varphi_a \in C$, $\varphi_b \notin C$. Define a recursive function f such that

$$\varphi_{f(x)} \simeq \begin{cases} \varphi_a & \text{if } \varphi_x \notin C \\ \varphi_b & \text{if } \varphi_x \in C \end{cases}$$

Then use the Fixed Point Theorem.)
Why doesn't this contradict Exercise 7?

Further Reading

Virtually all the theorems in this chapter were originally proved by Kleene, the Normal Form Theorem in particular in 1936; see his *Introduction to Metamathematics* for details of the history. Kleene's original computation predicate is called the "Kleene T-predicate"; our version of the Normal Form Theorem is based on one from G. Sacks via Robert W. Robinson.

The theorems in this chapter don't depend on the particular numbering we gave but only that there is *some* effective numbering of the partial recursive functions. The criteria for an acceptable numbering are discussed by Rogers in his book *Theory of Recursive Functions and Effective Computability* (exercises 2–10, 2–11, and 11–10) and by Odifreddi in *Classical Recursion Theory*, Chapter II.5.

For a more general discussion of the Fixed Point Theorem and extensions of it, see Odifreddi, especially Chapter II.2.

For the study of algorithmic classes, see Odifreddi, Chapter II.3 and Chapter V.