CHAPTER 3

Primitive Recursive Functions

1. Composition

We want to combine computable functions in such a way that the output of one becomes an input to another. In the simplest case we combine functions f and g to obtain the function

$$h(x) = f(g(x)).$$

More generally, for functions of several variables:

Definition. Let f be a function of k variables and let $g_1, \ldots, g_k$ be functions of n variables. Let

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)).$$

Then h is said to be obtained from f and $g_1, \ldots, g_k$ by *composition.*

Of course, the functions $f, g_1, \ldots, g_k$ need not be total. $h(x_1, \ldots, x_n)$ will be defined when all of $z_1 = g_1(x_1, \ldots, x_n), \ldots, z_k = g_k(x_1, \ldots, x_n)$ are defined and also $f(z_1, \ldots, z_k)$ is defined.

Using macros it is very easy to prove

Theorem 1.1. If h is obtained from the (partially) computable functions f, $g_1, \ldots, g_k$ by composition, then h is (partially) computable.

The word "partially" is placed in parentheses in order to assert the correctness of the statement with the word included or omitted in both places.

Proof. The following program obviously computes h:

$$\begin{aligned} Z_1 &\leftarrow g_1(X_1, \ldots, X_n) \\ &\vdots \\ Z_k &\leftarrow g_k(X_1, \ldots, X_n) \\ Y &\leftarrow f(Z_1, \ldots, Z_k) \end{aligned}$$

If $f, g_1, \ldots, g_k$ are not only partially computable but are also total, then so is h. ■

By Section 4 of Chapter 2, we know that x, $x + y$, $x \cdot y$, and $x - y$ are partially computable. So by Theorem 1.1 we see that $2x = x + x$ and $4x^2 = (2x) \cdot (2x)$ are computable. So are $4x^2 + 2x$ and $4x^2 - 2x$. Note that $4x^2 - 2x$ is total, although it is obtained from the nontotal function $x - y$ by composition with $4x^2$ and $2x$.

2. Recursion

Suppose k is some fixed number and

$$\begin{aligned} h(0) &= k, \\ h(t + 1) &= g(t, h(t)), \end{aligned} \tag{2.1}$$

where g is some given *total* function of two variables. Then h is said to be obtained from g by *recursion.*

Theorem 2.1. Let h be obtained from g as in (2.1), and let g be computable. Then h is also computable.

Proof. We first note that the constant function $f(x) = k$ is computable; in fact, it is computed by the program

$$\left.\begin{array}{l} Y \leftarrow Y + 1 \\ Y \leftarrow Y + 1 \\ \vdots \\ Y \leftarrow Y + 1 \end{array}\right\} k \text{ lines}$$

Hence we have available the macro $Y \leftarrow k$. The following is a program which computes $h(x)$:

$$\begin{array}{ll} & Y \leftarrow k \\ [A] & \text{IF } X = 0 \text{ GOTO } E \\ & Y \leftarrow g(Z, Y) \\ & Z \leftarrow Z + 1 \\ & X \leftarrow X - 1 \\ & \text{GOTO } A \end{array}$$

To see that this program does what it is supposed to do, note that if Y has the value $h(z)$ before executing the instruction labeled A, then it has the value $g(z, h(z)) = h(z + 1)$ after executing the instruction $Y \leftarrow g(Z, Y)$. Since Y is initialized to $k = h(0)$, Y successively takes on the values $h(0)$, $h(1), \ldots, h(x)$ and then terminates. ■

A slightly more complicated kind of recursion is involved when we have

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n), \\ h(x_1, \ldots, x_n, t+1) &= g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n). \end{aligned} \tag{2.2}$$

Here the function h of $n+1$ variables is said to be obtained by *recursion* from the total functions f (of n variables) and g (of $n+2$ variables). The recursion (2.2) is just like (2.1) except that parameters $x_1, \ldots, x_n$ are involved. Again we have

Theorem 2.2. Let h be obtained from f and g as in (2.2) and let f, g be computable. Then h is also computable.

Proof. The proof is almost the same as for Theorem 2.1. The following program computes $h(x_1, \ldots, x_n, x_{n+1})$:

$$\begin{array}{ll} & Y \leftarrow f(X_1, \ldots, X_n) \\ [A] & \text{IF } X_{n+1} = 0 \text{ GOTO } E \\ & Y \leftarrow g(Z, Y, X_1, \ldots, X_n) \\ & Z \leftarrow Z + 1 \\ & X_{n+1} \leftarrow X_{n+1} - 1 \\ & \text{GOTO } A \end{array}$$

■

3. PRC Classes

So far we have considered the operations of composition and recursion. Now we need some functions on which to get started. These will be

$$s(x) = x + 1,$$

$$n(x) = 0,$$

and the *projection functions*

$$u_i^n(x_1, \ldots, x_n) = x_i, \qquad 1 \leq i \leq n.$$

[For example, $u_3^4(x_1, x_2, x_3, x_4) = x_3$.] The functions s, n, and u_i^n are called the *initial functions.*

Definition. A class of total functions $\mathscr{C}$ is called a *PRC*[1] *class* if

1. the initial functions belong to $\mathscr{C}$,
2. a function obtained from functions belonging to $\mathscr{C}$ by either composition or recursion also belongs to $\mathscr{C}$.

[1] This is an abbreviation for "primitive recursively closed."

Then we have

Theorem 3.1. The class of computable functions is a PRC class.

Proof. By Theorems 1.1, 2.1, and 2.2, we need only verify that the initial functions are computable.

Now this is obvious; $s(x) = x + 1$ is computed by

$$Y \leftarrow X + 1$$

$n(x)$ is computed by the empty program, and $u_i^n(x_1, \ldots, x_n)$ is computed by the program

$$Y \leftarrow X_i$$ ■

Definition. A function is called *primitive recursive* if it can be obtained from the initial functions by a finite number of applications of composition and recursion.

It is obvious from this definition that

Corollary 3.2. The class of primitive recursive functions is a PRC class.

Actually we can say more:

Theorem 3.3. A function is primitive recursive if and only if it belongs to every PRC class.

Proof. If a function belongs to every PRC class, then, in particular, by Corollary 3.2, it belongs to the class of primitive recursive functions.

Conversely let a function f be a primitive recursive function and let $\mathscr{C}$ be some PRC class. We want to show that f belongs to $\mathscr{C}$. Since f is a primitive recursive function, there is a list $f_1, f_2, \ldots, f_n$ of functions such that $f_n = f$ and each f_i in the list is either an initial function or can be obtained from preceding functions in the list by composition or recursion. Now the initial functions certainly belong to the PRC class $\mathscr{C}$. Moreover the result of applying composition or recursion to functions in $\mathscr{C}$ is again a function belonging to $\mathscr{C}$. Hence each function in the list $f_1, \ldots, f_n$ belongs to $\mathscr{C}$. Since $f_n = f$, f belongs to $\mathscr{C}$. ■

Corollary 3.4. Every primitive recursive function is computable.

Proof. By the theorem just proved, every primitive recursive function belongs to the PRC class of computable functions. ■

In Chapter 13 we shall show how to obtain a computable function which is not primitive recursive. Hence it will follow that the set of primitive recursive functions is a proper subset of the set of computable functions.

4. Some Primitive Recursive Functions

We proceed to make a short list of primitive recursive functions. Being primitive recursive, they are also computable.

1. $x + y$.

To see that this is primitive recursive, we have to show how to obtain this function from the initial functions using only the operations of composition and recursion.

If we write $f(x, y) = x + y$, we have the recursion equations

$$f(x, 0) = x,$$
$$f(x, y + 1) = f(x, y) + 1.$$

We can rewrite these equations as

$$f(x, 0) = u_1^1(x),$$
$$f(x, y + 1) = s(u_2^3(y, f(x, y), x)).$$

The functions $u_1^1(x)$, $u_2^3(x_1, x_2, x_3)$, and $s(x)$ are primitive recursive functions; in fact, they are initial functions. Also, the function $s(u_2^3(x_1, x_2, x_3))$ is a primitive recursive function, since it is obtained by composition of primitive recursive functions. Thus, the above is a valid application of the operation of recursion to primitive recursive functions. Hence, $f(x, y) = x + y$ is primitive recursive.

Of course we already knew that $x + y$ was a computable function. So we have only obtained the additional information that it is in fact primitive recursive.

2. $x \cdot y$.

The recursion equations for $h(x, y) = x \cdot y$ are

$$h(x, 0) = 0,$$
$$h(x, y + 1) = h(x, y) + x.$$

This can be rewritten

$$h(x, 0) = n(x),$$
$$h(x, y + 1) = f(u_2^3(y, h(x, y), x), u_3^3(y, h(x, y), x)).$$

Here

$n(x)$ is the zero function,

$f(x_1, x_2)$ is $x_1 + x_2$,

and

$$u_2^3(x_1, x_2, x_3), u_3^3(x_1, x_2, x_3) \text{ are projection functions.}$$

Notice that the functions $n(x)$, $u_2^3(x_1, x_2, x_3)$, and $u_3^3(x_1, x_2, x_3)$ are all primitive recursive functions, since they are all initial functions. We have just shown that $f(x_1, x_2) = x_1 + x_2$ is primitive recursive. Thus, the function

$$f(u_2^3(x_1, x_2, x_3), u_3^3(x_1, x_2, x_3))$$

is primitive recursive since it is obtained by composition of primitive recursive functions. Finally, we conclude that

$$h(x, y) = x \cdot y$$

is primitive recursive.

3. $x!$.

The recursion equations are

$$0! = 1,$$
$$(x + 1)! = x! \cdot s(x).$$

More precisely, $x! = h(x)$, where

$$h(0) = 1,$$
$$h(t + 1) = g(t, h(t)),$$

and

$$g(x_1, x_2) = s(x_1) \cdot x_2.$$

Finally, g is primitive recursive because

$$g(x_1, x_2) = s(u_1^2(x_1, x_2)) \cdot u_2^2(x_1, x_2)$$

and multiplication is already known to be primitive recursive.

In the examples that follow, we leave it to the reader to check that the recursion equations can be put in the precise form called for by the definition of the operation of recursion.

4. x^y.

The recursion equations are

$$x^0 = 1,$$
$$x^{y+1} = x^y \cdot x.$$

Note that these equations assign the value 1 to the "indeterminate" 0^0.

5. $p(x)$.

The *predecessor function* $p(x)$ is defined as follows:

$$p(x) = \begin{cases} x - 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

It corresponds to the instruction in our programming language $X \leftarrow X - 1$.

The recursion equations for $p(x)$ are simply

$$p(0) = 0,$$

$$p(t + 1) = t.$$

Hence, $p(x)$ is primitive recursive.

6. $x \dot{-} y$.

The function $x \dot{-} y$ is defined as follows:

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y. \end{cases}$$

This function should not be confused with the function $x - y$, which is undefined if $x < y$. In particular, $x \dot{-} y$ is total, while $x - y$ is not.

We show that $x \dot{-} y$ is primitive recursive by displaying the recursion equations:

$$x \dot{-} 0 = x,$$

$$x \dot{-} (t + 1) = p(x \dot{-} t).$$

7. $|x - y|$.

The function $|x - y|$ is defined as the absolute value of the difference between x and y. It can be expressed simply as

$$|x - y| = (x \dot{-} y) + (y \dot{-} x)$$

and thus is primitive recursive.

8. $\alpha(x)$.

The function $\alpha(x)$ is defined as

$$\alpha(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0. \end{cases}$$

$\alpha(x)$ is primitive recursive since

$$\alpha(x) = 1 \dot{-} x.$$

Or we can simply write the recursion equations:

$$\alpha(0) = 1,$$

$$\alpha(t + 1) = 0.$$

Exercises

1. Show that for each k, the function $f(x) = k$ is primitive recursive.

2. Prove that if $f(x)$ and $g(x)$ are primitive recursive functions, so is $f(x) + g(x)$.

3.* (a) Let $E(x) = 0$ if x is even, $E(x) = 1$ if x is odd. Show that $E(x)$ is primitive recursive.

(b) Let $H(x) = x/2$ if x is even, $(x - 1)/2$ if x is odd. Show that H is primitive recursive.

4.* Let $g(x)$ be a primitive recursive function and let $f(0, x) = g(x)$, $f(n + 1, x) = f(n, f(n, x))$. Prove that $f(n, x)$ is primitive recursive.

5.* Let $f(0) = 0, f(1) = 1, f(2) = 2^2, f(3) = 3^{3^3} = 3^{27}$, etc. In general, $f(n)$ is written as a stack n high, of n's as exponents. Show that f is primitive recursive.

5. Primitive Recursive Predicates

We recall from Chapter 1, Section 4, that predicates or Boolean-valued functions are simply total functions whose values are 0 or 1. (We have identified 1 with TRUE and 0 with FALSE.) Thus we can speak without further ado of primitive recursive predicates.

We continue our list of primitive recursive functions, including some that are predicates.

9. $x = y$.

The predicate $x = y$ is defined as 1 if the values of x and y are the same and 0 otherwise. Thus we wish to show that the function

$$d(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

is primitive recursive. This follows immediately from the equation

$$d(x, y) = \alpha(|x - y|).$$

10. $x \leq y$.

This predicate is simply the primitive recursive function $\alpha(x \dot{-} y)$.

Theorem 5.1. Let $\mathscr{C}$ be a PRC class. If P, Q are predicates which belong to $\mathscr{C}$, then so are $\sim P$, $P \vee Q$, and $P \,\&\, Q$.[2]

[2] See Chapter 1, Section 4.

Proof. Since $\sim P = \alpha(P)$, it follows that $\sim P$ belongs to $\mathscr{C}$. (α was defined in Section 4, item 8.)

Also, we have

$$P \,\&\, Q = P \cdot Q,$$

so that $P \,\&\, Q$ belongs to $\mathscr{C}$.

Finally, the De Morgan law,

$$P \vee Q \Leftrightarrow \sim(\sim P \,\&\, \sim Q)$$

shows, using what we have already done, that $P \vee Q$ belongs to $\mathscr{C}$. ■

A result like Theorem 5.1 which refers to PRC classes can be applied to the two classes we have shown to be PRC. That is, taking $\mathscr{C}$ to be the class of all primitive recursive functions, we have

Corollary 5.2. If P, Q are primitive recursive predicates, then so are $\sim P$, $P \vee Q$, and $P \,\&\, Q$.

Similarly taking $\mathscr{C}$ to be the class of all computable functions, we have

Corollary 5.3. If P, Q are computable predicates, then so are $\sim P$, $P \vee Q$, and $P \,\&\, Q$.

As a simple example we have

11. $x < y$.

We can write

$$x < y \quad \Leftrightarrow \quad x \leq y \,\&\, \sim(x = y),$$

or more simply

$$x < y \quad \Leftrightarrow \quad \sim(y \leq x).$$

Theorem 5.4 (Definition by Cases). Let $\mathscr{C}$ be a PRC class. Let the functions g, h and the predicate P belong to $\mathscr{C}$. Let

$$f(x_1, \ldots, x_n) = \begin{cases} g(x_1, \ldots, x_n) & \text{if } P(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) & \text{otherwise.} \end{cases}$$

Then f belongs to $\mathscr{C}$.

This will be recognized as a version of the familiar "if ... then ..., else ..." statement.

Proof. The result is obvious because

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n) \cdot P(x_1, \ldots, x_n) + h(x_1, \ldots, x_n) \cdot \alpha(P(x_1, \ldots, x_n)).$$ ■

6. Iterated Operations and Bounded Quantifiers

Theorem 6.1 Let $\mathscr{C}$ be a PRC class. If $f(t, x_1, \ldots, x_n)$ belongs to $\mathscr{C}$, then so do the functions

$$g(y, x_1, \ldots, x_n) = \sum_{t=0}^{y} f(t, x_1, \ldots, x_n)$$

and

$$h(y, x_1, \ldots, x_n) = \prod_{t=0}^{y} f(t, x_1, \ldots, x_n).$$

A common error is to attempt to prove this by using mathematical induction on y. A little reflection reveals that such an argument by induction shows that

$$g(0, x_1, \ldots, x_n), g(1, x_1, \ldots, x_n), \ldots$$

all belong to $\mathscr{C}$, but not that the function $g(y, x_1, \ldots, x_n)$, one of whose arguments is y, belongs to $\mathscr{C}$.

We proceed with the correct proof.

Proof. We note the recursion equations

$$g(0, x_1, \ldots, x_n) = f(0, x_1, \ldots, x_n),$$
$$g(t + 1, x_1, \ldots, x_n) = g(t, x_1, \ldots, x_n) + f(t + 1, x_1, \ldots, x_n),$$

and recall that since $+$ is primitive recursive, it belongs to $\mathscr{C}$.

Similarly,

$$h(0, x_1, \ldots, x_n) = f(0, x_1, \ldots, x_n),$$
$$h(t + 1, x_1, \ldots, x_n) = h(t, x_1, \ldots, x_n) \cdot f(t + 1, x_1, \ldots, x_n).$$ ■

Sometimes we will want to begin the summation (or product) at 1 instead of 0. That is, we will want to consider

$$g(y, x_1, \ldots, x_n) = \sum_{t=1}^{y} f(t, x_1, \ldots, x_n)$$

or

$$h(y, x_1, \ldots, x_n) = \prod_{t=1}^{y} f(t, x_1, \ldots, x_n).$$

Then the initial recursion equations can be taken to be

$$g(0, x_1, \ldots, x_n) = 0,$$
$$h(0, x_1, \ldots, x_n) = 1,$$

with the equations for $g(t+1, x_1, \ldots, x_n)$ and $h(t+1, x_1, \ldots, x_n)$ as in the above proof. Note that we are implicitly defining a vacuous sum to be 0 and a vacuous product to be 1. With this understanding we have proved

Corollary 6.2. If $f(t, x_1, \ldots, x_n)$ belongs to the PRC class $\mathscr{C}$, then so do the functions

$$g(y, x_1, \ldots, x_n) = \sum_{t=1}^{y} f(t, x_1, \ldots, x_n)$$

and

$$h(y, x_1, \ldots, x_n) = \prod_{t=1}^{y} f(t, x_1, \ldots, x_n).$$

We have

Theorem 6.3. If the predicate $P(t, x_1, \ldots, x_n)$ belongs to some PRC class $\mathscr{C}$, then so do the predicates[3]

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n) \quad \text{and} \quad (\exists t)_{\leq y} P(t, x_1, \ldots, x_n).$$

Proof. We need only observe that

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n) \Leftrightarrow \left[\prod_{t=0}^{y} P(t, x_1, \ldots, x_n)\right] = 1$$

and

$$(\exists t)_{\leq y} P(t, x_1, \ldots, x_n) \Leftrightarrow \left[\sum_{t=0}^{y} P(t, x_1, \ldots, x_n)\right] \neq 0. \qquad \blacksquare$$

Actually for the universal quantifier it would even have been correct to write the equation

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n) = \prod_{t=0}^{y} P(t, x_1, \ldots, x_n).$$

Sometimes in applying Theorem 6.3 we want to use the quantifier

$$(\forall t)_{<y} \quad \text{or} \quad (\exists t)_{<y}.$$

That the theorem is still valid is clear from the relations

$$(\exists t)_{<y} P(t, x_1, \ldots, x_n) \Leftrightarrow (\exists t)_{\leq y}[t \neq y \,\&\, P(t, x_1, \ldots, x_n)],$$

$$(\forall t)_{<y} P(t, x_1, \ldots, x_n) \Leftrightarrow (\forall t)_{\leq y}[t = y \vee P(t, x_1, \ldots, x_n)].$$

We continue our list of examples:

[3] See Chapter 1, Section 5.

12. $y \mid x$.

This is the predicate "y is a divisor of x." For example,

$$3 \mid 12 \qquad \text{is true}$$

while

$$3 \mid 13 \qquad \text{is false.}$$

The predicate is primitive recursive since

$$y \mid x \iff (\exists t)_{\leq x}(y \cdot t = x).$$

13. Prime(x).

The predicate "x is a prime" is primitive recursive since

$$\text{Prime}(x) \Leftrightarrow x > 1 \,\&\, (\forall t)_{\leq x}\{t = 1 \vee t = x \vee \sim(t \mid x)\}.$$

(A number is a *prime* if it is greater than 1 and it has no divisors other than 1 and itself.)

Exercises

1. Let $f(x) = 2x$ if x is a perfect square; $f(x) = 2x + 1$ otherwise. Show that f is primitive recursive.

2. Let $\sigma(x)$ be the sum of the divisors of x if $x \neq 0$; $\sigma(0) = 0$ [e.g., $\sigma(6) = 1 + 2 + 3 + 6 = 12$]. Show that $\sigma(x)$ is primitive recursive.

3. Let $\pi(x)$ be the number of primes that are $\leq x$. Show that $\pi(x)$ is primitive recursive.

4. Let SQSM(x) be true if x is the sum of two perfect squares; false otherwise. Show that SQSM(x) is primitive recursive.

7. Minimalization

Let $P(t, x_1, \ldots, x_n)$ belong to some given PRC class $\mathscr{C}$. Then by Theorem 6.1, the function

$$g(y, x_1, \ldots, x_n) = \sum_{u=0}^{y} \prod_{t=0}^{u} \alpha(P(t, x_1, \ldots, x_n))$$

also belongs to $\mathscr{C}$. (Recall that the primitive recursive function α was defined in Section 4.) Let us analyze this function g. Suppose for definiteness that for some value of $t_0 \leq y$,

$$P(t, x_1, \ldots, x_n) = 0 \qquad \text{for} \quad t < t_0,$$

but

$$P(t_0, x_1, \ldots, x_n) = 1,$$

i.e., that t_0 *is the least value of* $t \leq y$ *for which* $P(t, x_1, \ldots, x_n)$ *is true.* Then

$$\prod_{t=0}^{u} \alpha(P(t, x_1, \ldots, x_n)) = \begin{cases} 1 & \text{if } u < t_0 \\ 0 & \text{if } u \geq t_0. \end{cases}$$

Hence,

$$g(y, x_1, \ldots, x_n) = \sum_{u < t_0} 1 = t_0,$$

so that $g(y, x_1, \ldots, x_n)$ is the least value of t for which $P(t, x_1, \ldots, x_n)$ is true. Now, we define

$$\min_{t \leq y} P(t, x_1, \ldots, x_n) = \begin{cases} g(y, x_1, \ldots, x_n) & \text{if } (\exists t)_{\leq y} P(t, x_1, \ldots, x_n) \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\min_{t \leq y} P(t, x_1, \ldots, x_n)$ *is the least value of* $t \leq y$ *for which* $P(t, x_1, \ldots, x_n)$ *is true, if such exists; otherwise it assumes the* (default) *value* 0. Using Theorems 5.4 and 6.3, we have

Theorem 7.1. If $P(t, x_1, \ldots, x_n)$ belongs to some PRC class $\mathscr{C}$ and $f(y, x_1, \ldots, x_n) = \min_{t \leq y} P(t, x_1, \ldots, x_n)$, then f also belongs to $\mathscr{C}$.

The operation "$\min_{t \leq y}$" is called *bounded minimalization.*

Continuing our list:

14. $\lfloor x/y \rfloor$

$\lfloor x/y \rfloor$ is the "integer part" of the quotient x/y. For example, $\lfloor 7/2 \rfloor = 3$ and $\lfloor 2/3 \rfloor = 0$. The equation

$$\lfloor x/y \rfloor = \min_{t \leq x} [(t + 1) \cdot y > x]$$

shows that $\lfloor x/y \rfloor$ is primitive recursive. Note that according to this equation, we are taking $\lfloor x/0 \rfloor = 0$.

15. $R(x, y)$.

$R(x, y)$ is the *remainder* when x is divided by y. Since

$$\frac{x}{y} = \lfloor x/y \rfloor + \frac{R(x, y)}{y},$$

we can write

$$R(x, y) = x \dot{-} (y \cdot \lfloor x/y \rfloor),$$

so that $R(x, y)$ is primitive recursive. [Note that $R(x, 0) = x$.]

16. p_n.

Here, for $n > 0$, p_n is the nth prime number (in order of size). So that p_n be a total function, we set $p_0 = 0$. Thus, $p_0 = 0$, $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, etc.

Consider the recursion equations

$$p_0 = 0,$$
$$p_{n+1} = \min_{t \leq p_n! + 1} [\text{Prime}(t) \,\&\, t > p_n].$$

To see that these equations are correct we must verify the inequality

$$p_{n+1} \leq (p_n)! + 1. \tag{7.1}$$

To do so note that for $0 < i \leq n$ we have

$$\frac{(p_n)! + 1}{p_i} = K + \frac{1}{p_i},$$

where K is an integer. Hence $(p_n)! + 1$ is not divisible by any of the primes $p_1, p_2, \ldots, p_n$. So, either $(p_n)! + 1$ is itself a prime or it is divisible by a prime $> p_n$. In either case there is a prime q such that $p_n < q \leq (p_n)! + 1$, which gives the inequality (7.1). (This argument is just Euclid's proof that there are infinitely many primes.)

Before we can confidently assert that p_n is a primitive recursive function, we need to justify the interleaving of the recursion equations with bounded minimalization. To do so, we first define the primitive recursive function

$$h(y, z) = \min_{t \leq z} [\text{Prime}(t) \,\&\, t > y].$$

Then we set

$$k(x) = h(x, x!+1),$$

another primitive recursive function. Finally, our recursion equations reduce to

$$p_0 = 0,$$
$$p_{n+1} = k(p_n),$$

so that we can conclude finally that p_n is a primitive recursive function.

It is worth noting that by using our various theorems (and appropriate macro expansions) we could now obtain explicitly a program of $\mathcal{S}$ which actually computes p_n. Of course the program obtained in this way would be extremely inefficient.

Now we want to discuss minimalization when there is no bound. We write

$$\min_y P(x_1, \ldots, x_n, y)$$

for the least value of y for which the predicate P is true *if there is one. If there is no value of y for which* $P(x_1, \ldots, x_n, y)$ *is true, then* $\min_y P(x_1, \ldots, x_n, y)$ *is undefined.* (Note carefully the difference with bounded minimalization.) Thus unbounded minimalization of a predicate can easily produce a function which is not total. For example,

$$x - y = \min_z [y + z = x]$$

is undefined for $x < y$. Now, as we shall see later, there are primitive recursive predicates $P(x, y)$ such that $\min_y P(x, y)$ is a total function which is *not* primitive recursive. However, we can prove

Theorem 7.2. If $P(x_1, \ldots, x_n, y)$ is a computable predicate and if

$$g(x_1, \ldots, x_n) = \min_y P(x_1, \ldots, x_n, y),$$

then g is a partially computable function.

Proof. The following program obviously computes g:

```
[A]   IF P(X_1, ..., X_n, Y) GOTO E
      Y ← Y + 1
      GOTO A
```

■

Exercises

1. Let $h(x)$ be the integer n such that $n \leq \sqrt{2}\, x < n + 1$. Show that $h(x)$ is primitive recursive.

2. Do the same when $h(x)$ is the integer n such that

$$n \leq (1 + \sqrt{2})\, x < n + 1.$$

3. p is called a *larger twin prime* if p and $p - 2$ are both primes. (5, 7, 13, 19 are larger twin primes.) Let $T(0) = 0$, $T(n) =$ the nth larger twin prime. It is widely believed, but has not been proved, that there are infinitely many larger twin primes. Assuming that this is true prove that $T(n)$ is computable.

4. Let $u(n)$ be the nth number in order of size which is the sum of two squares. Show that $u(n)$ is primitive recursive.

5. Let $R(x, t)$ be a primitive recursive predicate. Let

$$g(x, y) = \max_{t \leq y} R(x, t),$$

i.e., $g(x, y)$ is the largest value of $t \leq y$ for which $R(x, t)$ is true; if there is none, $g(x, y) = 0$. Prove that $g(x, y)$ is primitive recursive.

8. Pairing Functions and Gödel Numbers

In this section we shall study two convenient coding devices which use primitive recursive functions. The first is for coding pairs of numbers by single numbers, and the second is for coding lists of numbers.

We define the primitive recursive function

$$\langle x, y\rangle = 2^x(2y + 1) \dot{-} 1.$$

Note that $2^x(2y + 1) \neq 0$ so we can just as well omit the "dot" and write

$$\langle x, y\rangle = 2^x(2y + 1) - 1.$$

If z is any given number, there is a *unique* solution x, y to the equation

$$\langle x, y\rangle = z, \tag{8.1}$$

namely: x is the largest number such that $2^x|(z + 1)$, and y is then the solution of the equation

$$2y + 1 = (z + 1)/2^x;$$

this last equation has a (unique) solution because $(z + 1)/2^x$ must be odd. (The 2's have been "divided out.") Equation (8.1) thus defines functions

$$x = l(z), \qquad y = r(z).$$

Since Eq. (8.1) implies that $x, y < z + 1$ we have

$$l(z) \leq z, \qquad r(z) \leq z.$$

Hence we can write

$$l(z) = \min_{x \leq z} [(\exists y)_{\leq z}(z = \langle x, y\rangle)],$$
$$r(z) = \min_{y \leq z} [(\exists x)_{\leq z}(z = \langle x, y\rangle)],$$

so that $l(z)$, $r(z)$ are primitive recursive functions.

The definition of $l(z)$, $r(z)$ can be expressed by the statement

$$\langle x, y\rangle = z \quad \Leftrightarrow \quad x = l(z) \,\&\, y = r(z).$$

We summarize the properties of the functions $\langle x, y\rangle$, $l(z)$, and $r(z)$ in

Theorem 8.1 (Pairing Function Theorem). The functions $\langle x, y\rangle$, $l(z)$, and $r(z)$ have the following properties:

1. they are primitive recursive;
2. $l(\langle x, y\rangle) = x$, $r(\langle x, y\rangle) = y$;
3. $\langle l(z), r(z)\rangle = z$;
4. $l(z), r(z) \leq z$.

We next obtain primitive recursive functions which encode and decode arbitrary finite sequences of numbers. The method we use, first employed by Gödel, depends on the prime power decomposition of integers.

We define the *Gödel number* of the sequence $(a_1, \ldots, a_n)$ to be the number

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} p_i^{a_i}.$$

Thus, the Gödel number of the sequence (3, 1, 5, 4, 6) is

$$[3, 1, 5, 4, 6] = 2^3 \cdot 3^1 \cdot 5^5 \cdot 7^4 \cdot 11^6.$$

For each fixed n, the function $[a_1, \ldots, a_n]$ is clearly primitive recursive.

Gödel numbering satisfies the following uniqueness property:

Theorem 8.2. If $[a_1, \ldots, a_n] = [b_1, \ldots, b_n]$, then

$$a_i = b_i, \qquad i = 1, \ldots, n.$$

This result is an immediate consequence of the uniqueness of the factorization of integers into primes, sometimes referred to as the *unique factorization theorem* or the *fundamental theorem of arithmetic.* (For a proof, see any elementary number theory textbook.)

However, note that

$$[a_1, \ldots, a_n] = [a_1, \ldots, a_n, 0] \tag{8.2}$$

because $p_{n+1}^0 = 1$. This same result obviously holds for any finite number of zeros adjoined to the right end of a sequence. In particular, since

$$1 = 2^0 = 2^0 3^0 = 2^0 3^0 5^0 = \cdots,$$

it is natural to regard 1 as the Gödel number of the "empty" sequence of length 0, and it is useful to do so.

If one adjoins zero to the left end of a sequence, the Gödel number of the new sequence will not be the same as the Gödel number of the original sequence. For example,

$$[2, 3] = 2^2 \cdot 3^3 = 108,$$

and

$$[2, 3, 0] = 2^2 \cdot 3^3 \cdot 5^0 = 108,$$

but

$$[0, 2, 3] = 2^0 \cdot 3^2 \cdot 5^3 = 1125.$$

We will now define a primitive recursive function $(x)_i$ so that if

$$x = [a_1, \ldots, a_n],$$

then $(x)_i = a_i$. We set

$$(x)_i = \min_{t \leq x} (\sim p_i^{t+1} | x).$$

Note that $(x)_0 = 0$.

We shall also use the primitive recursive function

$$\mathrm{Lt}(x) = \min_{i \leq x} ((x)_i \neq 0 \,\&\, (\forall j)_{\leq x}(j \leq i \vee (x)_j = 0)).$$

Thus, if $x = 20 = 2^2 \cdot 5^1 = [2, 0, 1]$, then $(x)_3 = 1$, but $(x)_4 = (x)_5 = \cdots = (x)_{20} = 0$. So, $\mathrm{Lt}(20) = 3$. Also, $\mathrm{Lt}(0) = \mathrm{Lt}(1) = 0$. If $x > 1$, and $\mathrm{Lt}(x) = n$, then p_n divides x but no prime greater than p_n divides x. Note that $\mathrm{Lt}([a_1, \ldots, a_n]) = n$ if and only if $a_n \neq 0$.

We summarize the key properties of these primitive recursive functions:

Theorem 8.3 (Sequence Number Theorem).

a. $([a_1, \ldots, a_n])_i = \begin{cases} a_i & \text{if } 1 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$

b. $[(x)_1, \ldots, (x)_n] = x \quad \text{if} \quad n \geq \mathrm{Lt}(x).$

Exercises

1. Let $F(0) = 0$, $F(1) = 1$, $F(n + 2) = F(n + 1) + F(n)$. [$F(n)$ is the nth so-called Fibonacci number.] Prove that $F(n)$ is primitive recursive.

2. Let

$$h_1(x, 0) = f_1(x),$$

$$h_2(x, 0) = f_2(x),$$

$$h_1(x, t + 1) = g_1(x, h_1(x, t), h_2(x, t)),$$

$$h_2(x, t + 1) = g_2(x, h_1(x, t), h_2(x, t)).$$

Prove that if f_1, f_2, g_1, g_2 all belong to some PRC class $\mathscr{C}$, then h_1, h_2 do also.

3.* (Course-of-Values Recursion) (a) For $f(n)$ any function, we write

$$\tilde{f}(0) = 1, \tilde{f}(n) = [f(0), f(1), \ldots, f(n - 1)] \text{ if } n \neq 0.$$

Let

$$f(n) = g(\tilde{f}(n))$$

for all n. Show that if g is primitive recursive so is f.

(b) Let

$$f(0) = 1, \quad f(1) = 4, \quad f(2) = 6,$$

$$f(x + 3) = f(x) + f(x + 1)^2 + f(x + 2)^3.$$

Show that $f(x)$ is primitive recursive.