A characteristic of the natural numbers is that every natural number can be reached from 0 by applying the successor operation +1 finitely many times—any natural number is either 0 or the successor of the successor of 0.

One way to specify a function $h : \mathbb{N} \to \mathbb{N}$ that makes use of this fact is this: 
(a) specify what the value of $h$ is for argument 0, and 
(b) also specify how to, given the value of $h(x)$, compute the value of $h(x+1)$. For (a) tells us directly what $h(0)$ is, so $h$ is defined for 0. Now, using the instruction given by (b) for $x = 0$, we can compute $h(1) = h(0 + 1)$ from $h(0)$. Using the same instructions for $x = 1$, we compute $h(2) = h(1 + 1)$ from $h(1)$, and so on. For every natural number $x$, we’ll eventually reach the step where we define $h(x)$ from $h(x + 1)$, and so $h(x)$ is defined for all $x \in \mathbb{N}$.

For instance, suppose we specify $h : \mathbb{N} \to \mathbb{N}$ by the following two equations:

\[
\begin{align*}
h(0) &= 1 \\
h(x + 1) &= 2 \cdot h(x)
\end{align*}
\]

If we already know how to multiply, then these equations give us the information required for (a) and (b) above. By successively applying the second equation, we get that

\[
\begin{align*}
h(1) &= 2 \cdot h(0) = 2, \\
h(2) &= 2 \cdot h(1) = 2 \cdot 2, \\
h(3) &= 2 \cdot h(2) = 2 \cdot 2 \cdot 2, \\
& \vdots
\end{align*}
\]

We see that the function $h$ we have specified is $h(x) = 2^x$.

The characteristic feature of the natural numbers guarantees that there is only one function $h$ that meets these two criteria. A pair of equations like these is called a \textit{definition by primitive recursion} of the function $h$. It is so-called because we define $h$ “recursively,” i.e., the definition, specifically the second equation, involves $h$ itself on the right-hand-side. It is “primitive” because in defining $h(x + 1)$ we only use the value $h(x)$, i.e., the immediately preceding value. This is the simplest way of defining a function on $\mathbb{N}$ recursively.

We can define even more fundamental functions like addition and multiplication by primitive recursion. In these cases, however, the functions in question are 2-place. We fix one of the argument places, and use the other for the recursion. E.g., to define $\text{add}(x, y)$ we can fix $x$ and define the value first for $y = 0$ and then for $y + 1$ in terms of $y$. Since $x$ is fixed, it will appear on the left and on the right side of the defining equations.

\[
\begin{align*}
\text{add}(x, 0) &= x \\
\text{add}(x, y + 1) &= \text{add}(x, y) + 1
\end{align*}
\]

These equations specify the value of $\text{add}$ for all $x$ \textit{and} $y$. To find $\text{add}(2, 3)$, for instance, we apply the defining equations for $x = 2$, using the first to find...
add(2, 0) = 2, then using the second to successively find add(2, 1) = 2 + 1 = 3, 
add(2, 2) = 3 + 1 = 4, add(2, 3) = 4 + 1 = 5.

In the definition of add we used + on the right-hand-side of the second 
equation, but only to add 1. In other words, we used the successor function 
succ(z) = z + 1 and applied it to the previous value add(x, y) to define add(x, y + 1). So we can think of the recursive definition as given in terms of a single 
function which we apply to the previous value. However, it doesn’t hurt—
and sometimes is necessary—to allow the function to depend not just on the 
previous value but also on x and y. Consider:

\[
\begin{align*}
mult(x, 0) &= 0 \\
mult(x, y + 1) &= add(mult(x, y), x)
\end{align*}
\]

This is a primitive recursive definition of a function mult by applying the 
function add to both the preceding value mult(x, y) and the first argument x. It 
also defines the function mult(x, y) for all arguments x and y. For instance, 
mult(2, 3) is determined by successively computing mult(2, 0), mult(2, 1), mult(2, 2), 
and mult(2, 3):

\[
\begin{align*}
mult(2, 0) &= 0 \\
mult(2, 1) &= mult(2, 0 + 1) = add(mult(2, 0), 2) = add(0, 2) = 2 \\
mult(2, 2) &= mult(2, 1 + 1) = add(mult(2, 1), 2) = add(2, 2) = 4 \\
mult(2, 3) &= mult(2, 2 + 1) = add(mult(2, 2), 2) = add(4, 2) = 6
\end{align*}
\]

The general pattern then is this: to give a primitive recursive definition 
of a function h(x_0, \ldots, x_{k-1}, y), we provide two equations. The first defines the 
value of h(x_0, \ldots, x_{k-1}, 0) without reference to h. The second defines the value 
of h(x_0, \ldots, x_{k-1}, y + 1) in terms of h(x_0, \ldots, x_{k-1}, y), the other arguments x_0, 
\ldots, x_{k-1}, and y. Only the immediately preceding value of h may be used in 
that second equation. If we think of the operations given by the right-hand-
sides of these two equations as themselves being functions f and g, then the 
general pattern to define a new function h by primitive recursion is this:

\[
\begin{align*}
h(x_0, \ldots, x_{k-1}, 0) &= f(x_0, \ldots, x_{k-1}) \\
h(x_0, \ldots, x_{k-1}, y + 1) &= g(x_0, \ldots, x_{k-1}, y, h(x_0, \ldots, x_{k-1}, y))
\end{align*}
\]

In the case of add, we have k = 1 and f(x_0) = x_0 (the identity function), and 
g(x_0, y, z) = z + 1 (the 3-place function that returns the successor of its third 
argument):

\[
\begin{align*}
add(x_0, 0) &= f(x_0) = x_0 \\
add(x_0, y + 1) &= g(x_0, y, add(x_0, y)) = succ(add(x_0, y))
\end{align*}
\]

In the case of mult, we have f(x_0) = 0 (the constant function always returning 0) and g(x_0, y, z) = add(z, x_0) (the 3-place function that returns the sum
of its last and first argument):

\[
\begin{align*}
\text{mult}(x_0, 0) &= f(x_0) = 0 \\
\text{mult}(x_0, y + 1) &= g(x_0, y, \text{mult}(x_0, y)) = \text{add} (\text{mult}(x_0, y), x_0)
\end{align*}
\]

Photo Credits

Bibliography