rec.1 Other Recursions

cmp:rec:ore:sec

Using pairing and sequencing, we can justify more exotic (and useful) forms of primitive recursion. For example, it is often useful to define two functions simultaneously, such as in the following definition:

$$\begin{aligned} h_0(\vec{x}, 0) &= f_0(\vec{x}) \\ h_1(\vec{x}, 0) &= f_1(\vec{x}) \\ h_0(\vec{x}, y+1) &= g_0(\vec{x}, y, h_0(\vec{x}, y), h_1(\vec{x}, y)) \\ h_1(\vec{x}, y+1) &= g_1(\vec{x}, y, h_0(\vec{x}, y), h_1(\vec{x}, y)) \end{aligned}$$

This is an instance of *simultaneous recursion*. Another useful way of defining functions is to give the value of $h(\vec{x}, y+1)$ in terms of *all* the values $h(\vec{x}, 0)$, $\ldots$, $h(\vec{x}, y)$, as in the following definition:

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, y+1) &= g(\vec{x}, y, \langle h(\vec{x}, 0), \ldots, h(\vec{x}, y)\rangle). \end{aligned}$$

The following schema captures this idea more succinctly:

$$h(\vec{x}, y) = g(\vec{x}, y, \langle h(\vec{x}, 0), \ldots, h(\vec{x}, y-1)\rangle)$$

with the understanding that the last argument to g is just the empty sequence when y is 0. In either formulation, the idea is that in computing the "successor step," the function h can make use of the entire sequence of values computed so far. This is known as a *course-of-values* recursion. For a particular example, it can be used to justify the following type of definition:

$$h(\vec{x}, y) = \begin{cases} g(\vec{x}, y, h(\vec{x}, k(\vec{x}, y))) & \text{if } k(\vec{x}, y) < y \\ f(\vec{x}) & \text{otherwise} \end{cases}$$

In other words, the value of h at y can be computed in terms of the value of h at *any* previous value, given by k.

Problem rec.1. Define the remainder function $r(x, y)$ by course-of-values recursion. (If x, y are natural numbers and $y > 0$, $r(x, y)$ is the number less than y such that $x = z \times y + r(x, y)$ for some z. For definiteness, let's say that if $y = 0$, $r(x, 0) = 0$.)

You should think about how to obtain these functions using ordinary primitive recursion. One final version of primitive recursion is more flexible in that one is allowed to change the *parameters* (side values) along the way:

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, y+1) &= g(\vec{x}, y, h(k(\vec{x}), y)) \end{aligned}$$

This, too, can be simulated with ordinary primitive recursion. (Doing so is tricky. For a hint, try unwinding the computation by hand.)

Photo Credits

Bibliography