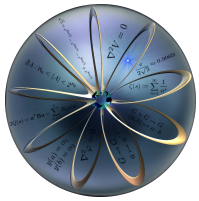

# planetmath.org

Math for the people, by the people.

## 1.10 Pattern matching and recursion

| | |
|---|---|
| Canonical name | 110PatternMatchingAndRecursion |
| Date of creation | 2013-11-13 18:36:23 |
| Last modified on | 2013-11-13 18:36:23 |
| Owner | PMBookProject (1000683) |
| Last modified by | rspuzio (6075) |
| Numerical id | 4 |
| Author | PMBookProject (6075) |
| Entry type | Feature |
| Classification | msc 03B15 |

The natural numbers introduce an additional subtlety over the types considered up until now. In the case of coproducts, for instance, we could define a function $f : A + B \to C$ either with the recursor:

$$f \equiv \mathsf{rec}_{A+B}(C, g_0, g_1)$$

or by giving the defining equations:

$$f(\mathsf{inl}(a)) \equiv g_0(a)$$
$$f(\mathsf{inr}(b)) \equiv g_1(b).$$

To go from the former expression of $f$ to the latter, we simply use the computation rules for the recursor. Conversely, given any defining equations

$$f(\mathsf{inl}(a)) \equiv \Phi_0$$
$$f(\mathsf{inr}(b)) \equiv \Phi_1$$

where $\Phi_0$ and $\Phi_1$ are expressions that may involve the variables $a$ and $b$ respectively, we can express these equations equivalently in terms of the recursor by using $\lambda$-abstraction:

$$f \equiv \mathsf{rec}_{A+B}(C, \lambda a.\, \Phi_0, \lambda b.\, \Phi_1).$$

In the case of the natural numbers, however, the "defining equations" of a function such as $\mathsf{double}$:

$$\mathsf{double}(0) \equiv 0 \tag{1}$$
$$\mathsf{double}(\mathsf{succ}(n)) \equiv \mathsf{succ}(\mathsf{succ}(\mathsf{double}(n))) \tag{2}$$

involve *the function* $\mathsf{double}$ *itself* on the right-hand side. However, we would still like to be able to give these equations, rather than (**??**), as the definition of $\mathsf{double}$, since they are much more convenient and readable. The solution is to read the expression "$\mathsf{double}(n)$" on the right-hand side of (**??**) as standing in for the result of the recursive call, which in a definition of the form $\mathsf{double} :\equiv \mathsf{rec}_{\mathbb{N}}(\mathbb{N}, c_0, c_s)$ would be the second argument of $c_s$.

More generally, if we have a "definition" of a function $f : \mathbb{N} \to C$ such as

$$f(0) \equiv \Phi_0$$
$$f(\mathsf{succ}(n)) \equiv \Phi_s$$

where $\Phi_0$ is an expression of type $C$, and $\Phi_s$ is an expression of type $C$ which may involve the variable $n$ and also the symbol "$f(n)$", we may translate it to a definition

$$f \equiv \mathsf{rec}_{\mathbb{N}}(C,\ \Phi_0,\ \lambda n.\, \lambda r.\, \Phi'_s)$$

where $\Phi'_s$ is obtained from $\Phi_s$ by replacing all occurrences of "$f(n)$" by the new variable $r$.

This style of defining functions by recursion (or, more generally, dependent functions by induction) is so convenient that we frequently adopt it. It is called definition by **pattern matching**. Of course, it is very similar to how a computer programmer may define a recursive function with a body that literally contains recursive calls to itself. However, unlike the programmer, we are restricted in what sort of recursive calls we can make: in order for such a definition to be re-expressible using the recursion principle, the function $f$ being defined can only appear in the body of $f(\mathsf{succ}(n))$ as part of the composite symbol "$f(n)$". Otherwise, we could write nonsense functions such as

$$f(0) \equiv 0$$
$$f(\mathsf{succ}(n)) \equiv f(\mathsf{succ}(\mathsf{succ}(n))).$$

If a programmer wrote such a function, it would simply call itself forever on any positive input, going into an infinite loop and never returning a value. In mathematics, however, to be worthy of the name, a *function* must always associate a unique output value to every input value, so this would be unacceptable.

This point will be even more important when we introduce more complicated inductive types in `http://planetmath.org/node/87578`Chapter 5,`http://planetmath.org` 6,`http://planetmath.org/node/87585`Chapter 11. Whenever we introduce a new kind of inductive definition, we always begin by deriving its induction principle. Only then do we introduce an appropriate sort of "pattern matching" which can be justified as a shorthand for the induction principle.