20 Primitive recursive functions

Note: this section is about natural numbers without reference to Peano Arithmetic.

(20.1) Definition *A* primitive recursive *function* $f : \mathbb{N}^k \to \mathbb{N}$ *is one which can be built from the following 'primitive' primitive recursive functions*

- *The* Zero function $Z(x) \equiv 0$,
- *The* Successor function $S(x) = x + 1$, *and*
- *The* Projection functions $P^n_k : \mathbb{N}^n \to \mathbb{N}$ *where* $1 \leqslant k \leqslant n$*:* $(x_1, \ldots, x_n) \mapsto x_k$

using the following operations

- *Substitution (aka composition): Given a function* $g : \mathbb{N}^k \to \mathbb{N}$ *and* k *functions* $h_j : \mathbb{N}^n \to \mathbb{N}$, $1 \leqslant j \leqslant k$*, the composite function* f *can be introduced as:*

$$f : \mathbb{N}^n \to \mathbb{N}; \quad \vec{x} \mapsto g(h_1(\vec{x}), \ldots, h_k(\vec{x}))$$

- *Primitive recursion: given a* $n-1$*-ary function* g *(if* $n = 1$ *then* g *is a constant) and a* $n+1$*-ary function* h*, the recursive formula defines* $f : \mathbb{N}^n \to \mathbb{N}$ *by primitive recursion from* g *and* h*: for any* $\vec{x} \in \mathbb{N}^{n-1}$

$$f(\vec{x}, 0) = g(\vec{x})$$
$$f(\vec{x}, r+1) = h(\vec{x}, r, f(\vec{x}, r))$$

20.1 Examples

Skipped (more than 2 pages).

20.2 Length-lexicographical

What we need for the study of computability is a theory of bitstrings, not of numbers. But it should be enough that bitstrings and numbers are interchangeable. That is, we want a bijection

$$\{0,1\}^* \to \mathbb{N}$$

The 'face value' of bitstrings is no use because it does not expect nor adjust to leading zeroes; also, because there is no number corresponding to the empty string λ.

We use $v(\alpha)$ for the face value of a string α, given that $\alpha \neq \lambda$.

The map chosen is to be called *length-lexicographical*. It defines the following ordering of bitstrings:

$$\lambda,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ 001,\ 010,\ 011,\ 100, \ldots$$

so the length-lex encoding of a string α is

$$\begin{cases} 0 & \text{if } \alpha = \lambda \\ 2^{|\alpha|} - 1 + v(\alpha) & \text{otherwise} \end{cases}$$

where $v(\alpha)$ is the face-value of α as a binary string.

To reduce confusion, in the remainder of this section, Greek letters $\alpha, \beta, \ldots$ will represent strings and $x, y, z, \ldots$ will represent natural numbers.

Of course $\alpha\beta$ is the concatenation of two strings and xy is the product of two numbers.

(20.2) Lemma *If α, β are nonempty then*

$$v(\alpha\beta) = 2^{|\beta|}v(\alpha) + v(\beta). \quad \blacksquare$$

The strings of length k, where $k > 0$, are those x such that

$$2^k - 1 \leqslant x \leqslant 2^{k+1} - 2$$

so

$$2^k \leqslant x + 1 \leqslant 2^{k+1} - 1$$

Note. All the functions mentioned in this subsection are primitive recursive. The proofs have been skipped.

Clearly, the length k of the string encoded by x is $\lfloor \log_2(x+1) \rfloor$, which is primitive recursive.

Now for concatenation: setting aside the case where α or β is empty, we have

$$\left(2^{|\alpha|} - 1 + v(\alpha), 2^{|\beta|} - 1 + v(\beta)\right) \mapsto 2^{|\alpha\beta|} - 1 + 2^{|\beta|}v(\alpha) + v(\beta).$$

We can untangle this as follows (x, y both nonzero).

Suppose $x = 2^{|\alpha|} - 1 + v(\alpha)$, $y = 2^{|\beta|} - 1 + v(\beta)$.

$$2^{|\alpha|} - 1 + v(\alpha) = x$$
$$v(\alpha) \leqslant 2^{|\alpha|} - 1$$
$$2^{|\alpha|} + v(\alpha) = x + 1$$
$$2^{|\alpha|} \leqslant x + 1 \leqslant 2^{|\alpha|+1} - 1$$
$$|\alpha| = \lfloor (\log_2(x+1) \rfloor$$

and the last is a primitive recursive function of x.

But then $x - 2^{|\alpha|} + 1$, i.e., $v(\alpha)$, is primitive recursive, and we can then construct the map

$$(x, y) \mapsto 2^{|\alpha\beta|} - 1 + v(\alpha\beta)$$

as a primitive recursive function of x and y.

Clearly

- If $x = 0$ then $x \cdot y = y$;
- if $y = 0$ then $x \cdot y = x$.