Natural Numbers

October 6, 2023

Contents

1 Expression

Unsigned numbers are the basis of numerical systems with higher expressiveness. They lead to signed integers, and from there we can construct a small data structure to express all real numbers. We will use church encodings to express natural numbers. This is a base 1 system of numeric expression. We can start by defining 0.

$$0_{\mathbb{N}} = \lambda s \lambda z.z$$

You'll notice this is α equivalent to the false function. This allows us to construct higher values in a way that allows us to take advantage of the natural number system to provide limited interval recursion, which will help us develop many more functions in the future. This however means that unlike most standards of programming, true does not evaluate to 1, or any other natural number or integer.

$$1_{\mathbb{N}} = \lambda s \lambda z.s(z)$$

$$2_{\mathbb{N}} = \lambda s \lambda z.s(s(z))$$

This pattern continues infinitely. We can use this for bounded recursion by simply passing a function followed by its initiating argument to a natural number.

$$3_{\mathbb{N}} \text{not}_b \text{T}$$

$$\rightarrow_\beta \text{not}_b(\text{not}_b(\text{not}_b(\text{T})))$$

$$\rightarrow_\beta \text{F}$$

2 Operations

2.1 Manipulation

The most basic functions for a base 1 numeric system are the successor and predecessor. The successor increments the value, and is quite simple.

$$\text{S}_{\mathbb{N}} = \lambda n \lambda s \lambda z.s(nsz)$$

$$\text{S}_{\mathbb{N}}(3_{\mathbb{N}}) \rightarrow_\beta 4_{\mathbb{N}}$$

The predecessor decrements the value and is a little more involved, we can use the bounded recursion trick however to achieve the n-1 successor of 0. This floors numbers at 0.

$$\text{P}_{\mathbb{N}} = \lambda n.n(\lambda p.\upsilon_2(\text{S}_{\mathbb{N}}(p\pi_1))(p\pi_1))(\upsilon_2 0_{\mathbb{N}} 0_{\mathbb{N}})\pi_2$$

$$\text{P}_{\mathbb{N}} 4_{\mathbb{N}} \rightarrow_\beta 3_{\mathbb{N}}$$

Using these two we can construct a set of standard numeric manipulation operators.

$$\text{add}_{\mathbb{N}} = \lambda n \lambda m.n \text{S}_{\mathbb{N}} m$$

$$\text{sub}_{\mathbb{N}} = \lambda n \lambda m.n \text{P}_{\mathbb{N}} m$$

$$\text{mul}_{\mathbb{N}} = \lambda n \lambda m.m(\text{S}_{\mathbb{N}} n)0_{\mathbb{N}}$$

$$\exp_{\mathbb{N}} = \lambda n \lambda m.m(\text{mul}_{\mathbb{N}} n)1_{\mathbb{N}}$$

2.2 Relational Operators

We'll start with a quick way to check if a natural number is 0.

$$0?_{\mathbb{N}} = \lambda n.n\mathrm{F}\mathrm{not}_b\mathrm{F}$$

Using this and subtraction, we have all we need to derive every standard relational operator on natural numbers.

$$\mathrm{lte}_{\mathbb{N}} = \lambda n\lambda m.0?_{\mathbb{N}}(\mathrm{sub}_{\mathbb{N}}nm)$$

$$\mathrm{eq}_{\mathbb{N}} = \lambda n\lambda m.\mathrm{and}_b(\mathrm{lte}_{\mathbb{N}}nm)(\mathrm{lte}_{\mathbb{N}}nm)$$

$$\mathrm{neq}_{\mathbb{N}} = \lambda n\lambda m.\mathrm{not}_b(\mathrm{eq}_{\mathbb{N}}nm)$$

$$\mathrm{lt}_{\mathbb{N}} = \lambda n\lambda m.\mathrm{and}_b(\mathrm{lte}_{\mathbb{N}}nm)(\mathrm{neq}_{\mathbb{N}}nm)$$

$$\mathrm{gte}_{\mathbb{N}} = \lambda n\lambda m.\mathrm{not}_b(\mathrm{lt}_{\mathbb{N}}nm)$$

$$\mathrm{gt}_{\mathbb{N}} = \lambda n\lambda m.\mathrm{not}_b(\mathrm{lte}_{\mathbb{N}}nm)$$

2.3 Simple Helper Functions

For defining real number systems later on, it will be useful to do other smaller comparisons.

$$\max_{\mathbb{N}} = \lambda x\lambda y.\mathrm{gt}_{\mathbb{N}}xyxy$$

$$\min_{\mathbb{N}} = \lambda x\lambda y.\mathrm{gt}_{\mathbb{N}}xyyx$$

3 Combinator Expansion

We've seen how natural numbers can be used to perform bounded recursion.

$$3_{\mathbb{N}}fx \rightarrow_\beta f(f(f(x)))$$

We can now also perform more useful recursive operations by modifying the Y-combinator. The following expanded y combinator recurrs a function f with initiating argument a until base case parameter b is returned.

$$\lambda f\lambda a\lambda b.(\lambda m\lambda x.\mathrm{eq}_{\mathbb{N}}bmm(x(fm)x))a(\lambda m\lambda x.\mathrm{eq}_{\mathbb{N}}bmm(x(fm)x)))$$

$$\rightarrow_\beta \lambda f\lambda a\lambda b.(\lambda g.gag)(\lambda m\lambda x.\mathrm{eq}_{\mathbb{N}}bmm(x(fm)x))$$

This is largely demonstrative and is not very useful with its limited conditional ability, so we can abstract it further to achieve a more useful function; here is a combinator which recurrs function f with initiating argument a until condition c is met.

$$\mathrm{comb} = \lambda f\lambda a\lambda c.(\lambda g.gag)(\lambda m\lambda x.cmm(x(fm)x))$$

It is important to note that c is a function which is conditional on the returned value of f per recursion, and that the final returned state of calling this will be the final state of f.

We can now define natural number division.

$$\begin{aligned}\mathrm{div}_{\mathbb{N}} = \quad & \lambda m\lambda n.(\lambda g.gn1_{\mathbb{N}}g)(\lambda y\lambda i\lambda f.\mathrm{gt}_{\mathbb{N}}ym(\\ & \quad \mathrm{P}_{\mathbb{N}}i\\ &)(\\ & \quad \mathrm{eq}_{\mathbb{N}}ymi(f(\mathrm{add}_{\mathbb{N}}yn)(S_{\mathbb{N}}i)f)\\ &))\end{aligned}$$

Because it will help us create a real number system, lets also create a function that determines if n divides m evenly using our expanded knowledge of combinators.

$$\begin{aligned}\mathrm{divides}_{\mathbb{N}} = \quad & \lambda n\lambda m.(\lambda g.gng)(\lambda y\lambda f.\mathrm{gt}_{\mathbb{N}}ym\mathrm{F}(\\ & \quad \mathrm{eq}_{\mathbb{N}}ym\mathrm{T}(f(\mathrm{add}_{\mathbb{N}}yn)f)\\ &))\end{aligned}$$

We can also define greatest common factor.

$$\mathrm{gcf}_{\mathbb{N}} = \quad \lambda m\lambda c.(\lambda g.gmg)(\lambda y\lambda f.\mathrm{divides}_{\mathbb{N}}ycy(f(\mathrm{P}_{\mathbb{N}}y)f))$$