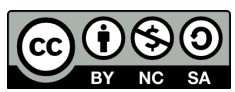
CDM

Primitive Recursion

Klaus Sutner

Carnegie Mellon University

Spring 2021



1 **Coding the World**

2 **Pairing Systems**

3 **Coding Systems**

4 ***Digression: Gödel's β Function**

Our primitive recursive programming language has one glaring defect: it only supports one data type, $\mathbb{N}$. There are no lists, trees, graphs, hash tables and so on, only natural numbers.

As it turns out, all these discrete structures can be obtained from just integers if we are able to express sequences $a_1, a_2, \ldots, a_n$ of numbers as a single number $\langle a_1, a_2, \ldots, a_n \rangle$.

This is obviously not meant as a practical programming idea, it is purely conceptual: natural numbers already suffice in principle, and the ability to compute with them means that other computation involving, say, list, are also possible.

We claim that any algorithm you will ever see, outside of a class dealing directly with logic and computability, is always primitive recursive. And, in fact, often trivially so.

There are two parts to this claim:

- All these algorithms operate on finitary data structures that can be coded as natural numbers, and
- given this coding, for input as well as output, the corresponding functions are always primitive recursive.

Of course, there is no actual theorem here, just an observation. I'd be most curious to hear about anything that might contradict this claim*.

*I will change my definition of RealWorld™

Leopold Kronecker, Semi-Constructivist

Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.

"Dear god" made the integers, everything else is the work of men.

Write $\mathbb{N}^\star$ for the set of all finite sequences of natural numbers and nil for the empty sequence.

To express a sequence $a_1, \ldots, a_n \in \mathbb{N}^\star$ as a single number $\langle a_1, \ldots, a_n \rangle$, we need a coding function, a polyadic injective map of the form

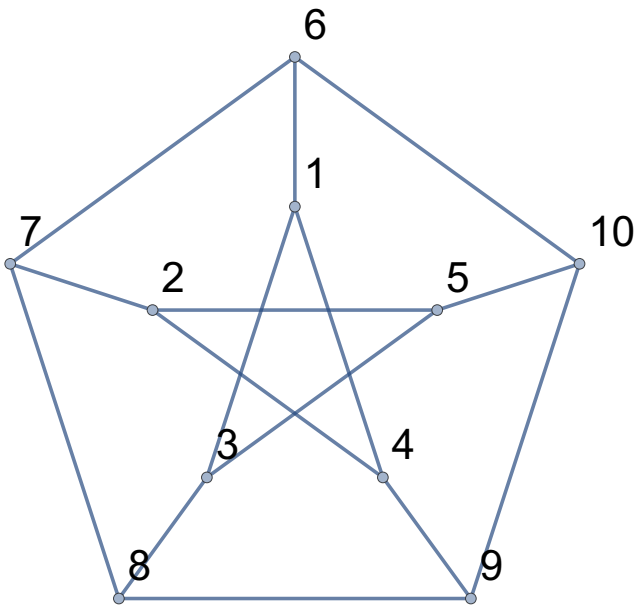
$$\langle . \rangle : \mathbb{N}^* \to \mathbb{N}$$

that allows us to decode: from $b = \langle a_1, \ldots, a_n \rangle$ we can recover n as well as all the a_i.

From the perspective of set theory this is trivial, but we interested in operations that are computationally cheap, at least primitive recursive.

We can now code any discrete structure as an integer by expressing it as a nested list of natural numbers, and then applying the coding function.

For example, the so-called Petersen graph on the left is given by its edge list, the nested list of naturals on the right.



$$((1,3),(1,4),(2,4),(2,5),(3,5),$$
$$(6,7),(7,8),(8,9),(9,10),(6,10),$$
$$(1,6),(2,7),(3,8),(4,9),(5,10))$$

In a coding system discussed below, the edges have code numbers

$$34, 66, 258, 132, 260, 1028, 520, 4104, 16400,$$
$$65568, 16448, 131136, 65664, 262400, 1049088$$

The code number of the last sequence is about

$$3.210742533937650 \times 10^{485597}$$

and has almost half a million digits.

This has nothing to do with the desing of efficient data structures, it is a tool to explore the power and limits of computation.

1 Coding the World

2 **Pairing Systems**

3 Coding Systems

4 *Digression: Gödel's β Function

Definition

A pairing system consists of

- an injective function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, the pairing function
- functions $\pi_i : \mathbb{N} \to \mathbb{N}$, $i = 1, 2$, the unpairing functions

where $\pi_i(\pi(x_1, x_2)) = x_i$ for all x_i.

The system is primitive recursive if the paring and unpairing functions as well as the range of the pairing function are all primitive recursive.

The unpairing functions are only of interest only on the range of π; we will generally assume that they are 0 everywhere else.

Note that this is not a problem for primitive recursive systems.

Note that any pairing function induces a total order on $\mathbb{N} \times \mathbb{N}$:

$$(a, b) \prec (a', b') \iff \pi(a, b) < \pi(a', b')$$

This provides a convenient method to construct pairing functions: find some reasonable ordering on $\mathbb{N} \times \mathbb{N}$ and then engineer a corresponding function.

We would like the function to be primitive recursive, so the ordering should be fairly straightforward.

For example, we could define the order to be

$$(a,b) \prec (a',b') \iff (a{+}b < a'{+}b') \vee \Big((a{+}b = a'{+}b' \wedge (a,b) <_p (a',b')\Big)$$

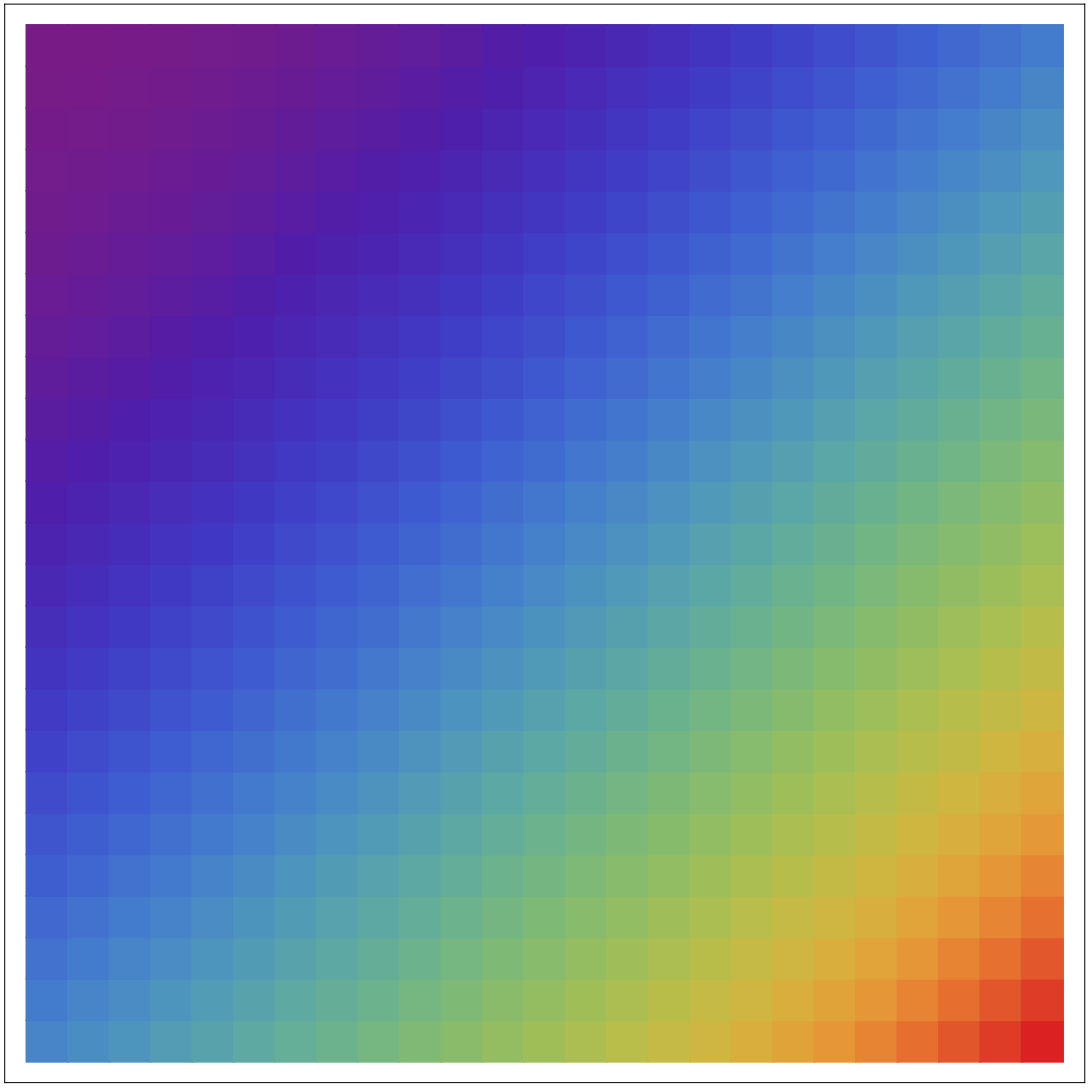
Here $<_p$ refers to the usual product order. So the first few pairs are

$$(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),\ldots$$

The corresponding pairing function is a simple quadratic polynomial:

$$\pi(x,y) = ((x+y)^2 + 3x + y)/2$$

Note that this function is actually a bijection.



Exercise

Explain the Cantor polynomial and show that function is indeed a bijection.

Exercise

Find simple descriptions for the corresponding unpairing functions.

Exercise

Show that the unpairing functions are primitive recursive.

Aside: Fueter-Pólya

A surprising theorem by Fueter and Pólya from 1923 states that, up to a swap of variables, this is the only quadratic polynomial that defines a bijection $\mathbb{N}^2 \leftrightarrow \mathbb{N}$.

The proof is rather difficult and uses the fact that e^a is transcendental for algebraic $a \neq 0$.

It is an open problem whether there are other bijections for higher degree polynomials.

Extra Credit.

Another Pairing Function

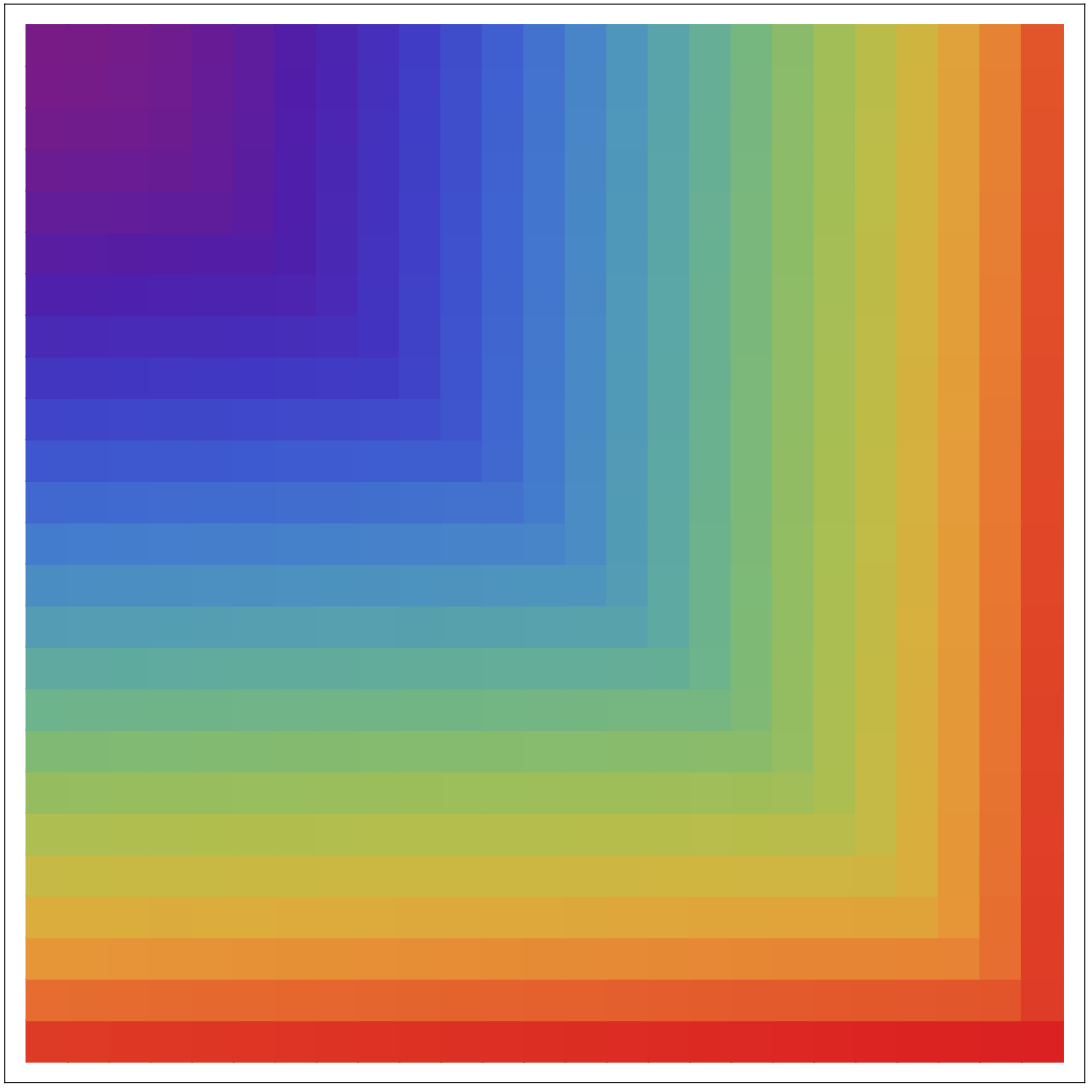
We could replace sum by max in the pair ordering:

$$(a,b) \prec (a',b') \iff (\max(a,b) < \max(a',b')) \vee \Big(\max(a,b) = \max(a',b') \wedge (a,b) <_p (a',b')\Big)$$

Again, $<_p$ refers to the usual product order. he first few pairs this time are

$$(0,0),(0,1),(1,0),(1,1),(0,2),(1,2),(2,0),(2,1),(2,2),(0,3),\ldots$$

Somewhat similar to the last pairing function, but the picture looks quite different.



Exercise

Find the pairing function for the last order.

Exercise

Then determine the corresponding unpairing functions.

Exercise

Show that the unpairing functions are primitive recursive.

The last two pairing functions are inspired by pair orderings. Then next one is based on basic arithmetic instead: decomposing a number into an even and on odd part.

$$\pi(x, y) = 2^x(2y + 1)$$

Hence the range of π is $\mathbb{N}_+$ (but not $\mathbb{N}$). The pairs this time look like so:

$$(0, 0), (1, 0), (0, 1), (2, 0), (0, 2), (1, 1), (0, 3), (3, 0), (0, 4), (1, 2), \ldots$$

Much less intuitive than the previous cases.

To see why this pairing function is still quite natural consider the numbers in binary. For example

$$\pi(5, 27) = 32 \cdot 55 = 1760 = 11011100000_2$$

In general, the binary expansion of $\pi(x, y)$ looks like so:

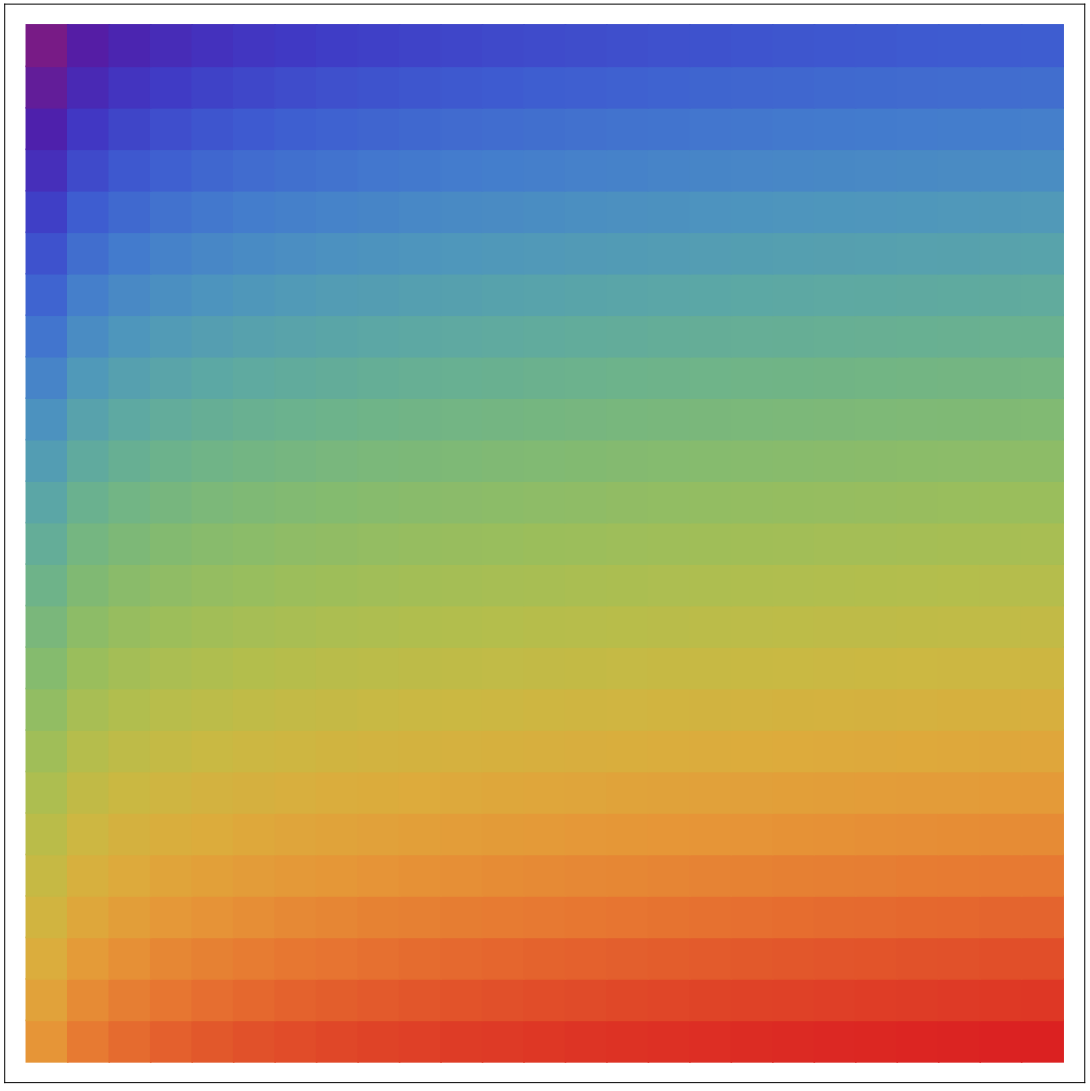
$$y_k y_{k-1} \dots y_0 \, 1 \, \underbrace{00 \dots 0}_{x}$$

where $y_k y_{k-1} \dots y_0$ is the standard binary expansion of y (y_k is the most significant digit).

Exercise

Find the corresponding unpairing functions.

Picture (log plot)



1 Coding the World

2 Pairing Systems

3 **Coding Systems**

4 *Digression: Gödel's β Function

Definition

A coding system consists of three functions

$$\langle . \rangle : \mathbb{N}^\star \to \mathbb{N}$$
$$\mathsf{len} : \mathbb{N} \to \mathbb{N}$$
$$\mathsf{dec} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

For $b = \langle a_1, a_2, \ldots, a_n \rangle$ we have $n = \mathsf{len}(b)$ and $\mathsf{dec}(b, i) = a_i$ for all $i \in [n]$.

$\langle . \rangle$ is polyadic and called the coding function, len is the length function, and dec is the decoding function. The range Seq of the coding function is the set of sequence numbers.

Seq may be equal to $\mathbb{N}$, but in general it will not be.

To be more explicit about the decoding process, suppose

$$b = \langle a_1, a_2, \ldots, a_n \rangle$$

is some sequence number. Then we can recover the length of the sequence via

$$\mathsf{len}(b) = n$$

and the actual entries via that extracts the components:

$$\mathsf{dec}(b, i) = a_i$$

for all $i = 0, \ldots, n-1$.

For simplicity we assume that len and dec are 0 outside of their relevant domain of definition.

Definition

A coding system is primitive recursive if the length and decoding functions, and the sequence numbers are all primitive recursive.

Note that the coding function itself is polyadic, and thus cannot be primitive recursive. In all interesting cases, the restrictions

$$\langle . \rangle : \mathbb{N}^n \to \mathbb{N}$$

will be primitive recursive, though.

So the challenge is to come up with well-behaved primitive recursive coding systems.

Exercise

Show how to check if a number is a sequence number given dec *and* len*.*

Suppose we have a pairing system. The first step is to extend the pairing function π to a map $\widehat{\pi}$ that is defined on all sequences of length at least 2:

$$\widehat{\pi} : \mathbb{N}^{\geq 2} \longrightarrow \mathbb{N}$$

This comes down to declaring π to be, say, right associative:

$$\widehat{\pi}(x_1, x_2) = \pi(x_1, x_2)$$
$$\widehat{\pi}(x_1, x_2, \ldots, x_k) = \pi(x_1, \widehat{\pi}(x_2, \ldots, x_k))$$

Note that this map is not injective: let $c = \pi(a, b)$, then $\widehat{\pi}(a, c) = \widehat{\pi}(a, a, b)$.

To avoid this issue, define

$$\begin{aligned}\langle \mathsf{nil} \rangle &:= \pi(0,0)\\ \langle a \rangle &:= \pi(1,a)\\ \langle a_1,\ldots,a_n \rangle &:= \pi(n,\widehat{\pi}(a_2,\ldots,a_n))\end{aligned}$$

Here are some sequence numbers for this particular coding function:

$$\begin{aligned}\langle 10 \rangle &= 1024\\ \langle 0,0,0 \rangle &= 7\\ \langle 1,2,3,4,5 \rangle &= 532754\end{aligned}$$

Lemma

$\langle . \rangle : \mathbb{N}^* \to \mathbb{N}$ *is injective.*

Proof. Suppose

$$\langle a_1, \ldots, a_1 \rangle = c = \langle b_1, \ldots, b_m \rangle$$

Since $\pi_1(c)$ is the length of the sequence we can conclude that $n = m$.

But then $\widehat{\pi}(a_1, \ldots, a_n) = \widehat{\pi}(b_1, \ldots, b_n)$ and it follows that $a_i = b_i$. □

Recall the even/odd pairing function

$$\pi(x, y) = 2^x \, (2y + 1)$$

The range here is $\mathbb{N}_+$, so we don't have a bijection. As it turns out, we can exploit this produce a rather elegant coding function:

$$\langle \mathsf{nil} \rangle \coloneqq 0$$

$$\langle a_1, \ldots, a_n \rangle \coloneqq \pi(a_1, \langle a_2, \ldots, a_n \rangle)$$

Lemma

$\langle . \rangle : \mathbb{N}^* \to \mathbb{N}$ *is bijective.*

Proof. Suppose

$$\langle a_1, \ldots, a_n \rangle = c = \langle b_1, \ldots, b_m \rangle$$

We may safely assume that $n \leq m$. If $n = 0$, then $c = 0$ and it follows that $m = 0$ because the range of π does not contain 0.

So suppose $0 < c$, $0 < n \leq m$. Since the range of π is all of $\mathbb{N}_+$ we have $a_1 = \pi_1(c) = b_1$, furthermore $\langle a_2, \ldots, a_n \rangle = \pi_2(c) = \langle b_2, \ldots, b_n \rangle$.

Done by induction. □

Here is a sequence number and its binary expansion:

$$\begin{aligned}\langle 2,3,5,1\rangle &= 20548\\ &= 1\underbrace{0}_{1}1\underbrace{00000}_{5}1\underbrace{000}_{3}1\underbrace{00}_{2}\end{aligned}$$

So the number of 1's (the digitsum) is just the length of the sequence, and the spacing between the 1's indicates the actual numerical values.

It follows that the coding function is injective and surjective, right?

Exercise

Show that the pairing function π and both unpairing functions $x = \pi_1(\pi(x, y))$ and $y = \pi_2(\pi(x, y))$ are primitive recursive.

Exercise

Show that the length and decoding functions len *and* dec *are primitive recursive.*

Exercise

Show that the coding function $\langle . \rangle$ is primitive recursive when restricted to inputs of fixed length.

1 Coding the World

2 Pairing Systems

3 Coding Systems

4 *Digression: Gödel's β Function

Traditionally, one uses 0-indexing for sequence numbers: $\langle a_0, \ldots, a_{n-1} \rangle$ so that the decoding function takes the form $\mathrm{dec}(b, i)$, $i < n$. Also, $\mathrm{dec}(b, i)$ is written $(b)_i$.

One neat application of sequence numbers is course-of-value recursion. First note that ordinary primitive recursion can be expressed in terms of sequence numbers like so:

$$\begin{aligned} f(x, \boldsymbol{y}) = z \iff \exists\, s \in \mathsf{Seq}\, \big(\, & \mathsf{len}(s) = x^+ \wedge \\ & (s)_0 = g(\boldsymbol{y}) \ \wedge \\ & \forall\, i < x\, ((s)_{i^+} = h(i, (s)_i, \boldsymbol{y})) \wedge \\ & (s)_x = z\, \big) \end{aligned}$$

Here x^+ is shorthand for $x + 1$. The sequence number s simply records all previous values of f.

Now consider the following function associated with f:

$$\widehat{f}(x, \boldsymbol{y}) := \langle f(0, \boldsymbol{y}), f(1, \boldsymbol{y}), \ldots, f(x, \boldsymbol{y}) \rangle$$

Lemma

f is primitive recursive iff $\widehat{f}$ is primitive recursive.

Thus, it is natural to generalize the primitive recursion scheme slightly by defining functions so that the value at x depends directly on all the previous values.

$$f(0, \boldsymbol{y}) = g(\boldsymbol{y})$$
$$f(x^+, \boldsymbol{y}) = H(x, \widehat{f}(x, \boldsymbol{y}), \boldsymbol{y})$$

Lemma

If g and H are primitive recursive then f is also primitive recursive.

This line of reasoning appeared first in Gödel's seminal 1931 paper establishing the incompleteness theorem.

For the sake of completeness, here is a brief description of Gödel's method.

The full sequence number machinery is not needed for course-of-value recursion. We can get away with a function

$$\beta(x, y, z) = x \bmod y(z + 1) + 1$$

Leaving off the parameters, let $f(0) = \alpha$, $f(x^+) = h(x, f(x))$. Then one can use β to express the recursion:

$$\begin{aligned} f(x) = z \iff \exists\, u, v \Big(& \beta(u, v, 0) = \alpha \wedge \\ & \forall\, i < x\, (\beta(u, v, i^+) = h(i, \beta(u, v, i))) \wedge \\ & \beta(u, v, x) = z \Big) \end{aligned}$$

β suffices to encode sequences. Here is an eloboration of this idea.

Lemma (Gödel)

There exists a primitive recursive function $\mathsf{dec} : \mathbb{N}^2 \to \mathbb{N}$ *such that*

$$\forall\, a_0, \ldots, a_{n-1}\, \exists\, a\, \forall\, i < n\, (a_i = \mathsf{dec}(a, i)).$$

So a is a potential code number for $a_0, \ldots, a_{n-1}$

Proof. Set

$$\mathsf{dec}(a, i) = \min\Big(x < a \mid \big(\pi_2(a)(\pi(x, i) + 1) + 1\big) \text{ divides } \pi_1(a) \Big)$$

Think of $a = \pi(u, v)$, then

$$\beta(u, v, \pi(x, i)) = 0$$

We need to establish the existence of the witness a.

Proof, contd.

Let $a_0, \ldots, a_{n-1}$ arbitrary and set

$$\begin{aligned} c &= \max\big(\, \pi(a_i, i) \mid i < n \,\big) \\ C &= (c-1)! \\ p &= \prod_{i<n} (C(\pi(a_i, i) + 1) + 1) \\ a &= \pi(p, C) \end{aligned}$$

Note that $\forall\, i < j < c\,(\, iC + 1, jC + 1$ coprime $)$.

But then

$$\begin{aligned} \mathsf{dec}(a, i) &= \min\big(\, x < a \mid C(\pi(x, i) + 1) + 1 \text{ divides } p \,\big) \\ &= \min\big(\, x < a \mid \beta(p, C, \pi(x, i)) = 0 \,\big) \end{aligned}$$

□

Definition

Define a coding function $\langle . \rangle$ by

$$\langle \boldsymbol{x} \rangle = \min\Big(a \mid \mathsf{dec}(a, 0) = n \wedge \forall\, i \in [n]\, (\mathsf{dec}(a, i) = a_i) \Big)$$

Then the length of a sequence is $\mathsf{dec}(a, 0)$. As usual, $\langle . \rangle$ is not primitive recursive, but nearly so:

- $\mathsf{Seq} = \{ \langle \boldsymbol{x} \rangle \mid \boldsymbol{x} \in \mathbb{N}^* \} \subseteq \mathbb{N}$ is primitive recursive.
- The restriction to $\mathbb{N}^n$ is primitive recursive.
- dec is primitive recursive.

Theorem (Kuznecov 1950)

The class of primitive recursive bijections on $\mathbb{N}$ is not closed under inverse.

Proof. Define the Ackermann-like function

$$\begin{aligned} B_0(x) &= 2x \\ B_{n+}(x) &= B_n^x(1) \\ B(x) &= B_x(x) \end{aligned}$$

It follows from monotonicity that the predicate "$B_n(x) = y$" is primitive recursive, uniformly in n, x, y.

Let R be the range of $B : \mathbb{N} \to \mathbb{N}$, so R is infinite, co-infinite and primitive recursive. Note that R is very sparse thanks to the insane growth of B.

Let H_X be the Hauptfunktion of $X \subseteq \mathbb{N}$ and define $f : \mathbb{N} \to \mathbb{N}$

$$f(x) = \begin{cases} 2\, H_R^{-1}(x) & \text{if } x \in R, \\ 2\, H_{\overline{R}}^{-1}(x) + 1 & \text{otherwise.} \end{cases}$$

Then f is an primitive recursive bijection, but f^{-1} fails to be primitive recursive.

Exercise

Prove that all these functions are indeed primitive recursive.

Exercise

Explain how to implement search in binary search trees as a primitive recursive operation.

Exercise

Come up with yet another coding function based on repeated application of a pairing function (make sure your method really works).