Creativistic Philosophy: Exploring the Limits of Formalization, #8[1]—Data and Gödel Numbers

Andreas Keller

(Revoy, 2025)[2]

[2] The cartoon, by David Revoy (see link in the references), shared here under the Creative Commons Attribution-ShareAlike 4.0 International licence, makes fun of the concept of AGI (Artificial General Intelligence), a topic discussed below at the end.

1. Digital Data

In the last installments, I moved away a bit from the mathematical considerations of the previous parts and included some philosophical thoughts. In the current installment, I plunge back into mathematics before I will return to philosophy later on, a few installments down the road (and briefly at the end of this one).

The data that are stored in, and processed by, computer systems (including smartphones and the internet and embedded systems in all kinds of devices) are of many different kinds. There are numbers and texts, pictures, sounds, videos, web pages, programs (like operating systems and "apps"), measurement values, control data for machines (including "robots"), and many more.

2. Gödel Numbers

It might therefore come as a surprise that some of the formal models of the concept of computation, or of algorithms, are limited to processing natural numbers. For example, the so-called "recursive functions" are defined on natural numbers only.[3]

However, nothing is lost by making such a restriction, because it is possible to code data of any kind as natural numbers in such a way that the coding operation (from data to natural number) and the corresponding decoding operation are both computable. This idea was used by Kurt Gödel in his famous incompleteness proofs. Initially, he used it to map formulas of a logical formalism to numbers and vice versa. For this reason, such mappings where later called "Gödelizations" in his honor, and the resulting numbers are called "Gödel numbers." This idea can be generalized to data of any kind.

3. An Example of Gödelization

Gödlizations can be constructed in many different ways. All you need is a computable function that maps data to natural numbers and for which the inverse function that maps the numbers back to the original data is also computable. In this section, we show a simple example of how this can be achieved.

Data in computers is usually represented as sequences of 0s and 1s. No matter what data type, if we look at the content of files stored on storage media or loaded into a computer's memory, all data is stored in binary form, i.e. as sequences of two different values that are usually denoted as "0" and "1."

[3] (Dean, 2022)

Such sequences are not numbers by themselves, but they can easily be mapped to natural numbers. Instead of decimal numbers, which are based on 10 digits and powers of 10 (1, 10, 100, 1000 etc.) as place values, we can use the binary number system, which is based on just two digits (0 and 1) and powers of 2 (1, 2, 4, 8, etc.) as place values[4] for this purpose.

Since we want to distinguish between data consisting of sequences that start with different numbers of 0s, like "001" and "01," we do not use the binary sequences as numbers directly. Instead we can map them onto numbers by placing a 1 in front ("1001" and "101" in the example).

Moreover, we want every natural number to occur as a code for a sequence of digits. If the "sequence" we are starting with is just a single "0," putting a "1" in front would yield "10," which is the binary representation of the number 2, so 1 would not occur as a Gödel number. To solve this problem, we simply subtract 1. Let us look at the following table to make this clearer.

Binary sequence (data)	=> Put "1" in front	=> Interpret as binary number	=> subtract 1	=> corresponding Decimal number
<= remove leading "1"	<= interpret as string of digits	<= add 1	<= turn into binary number	Gödel number
0	10	10	1	1
1	11	11	10	2
00	100	100	11	3
01	101	101	100	4
10	110	110	101	5
11	111	111	110	6
000	1000	1000	111	7
001	1001	1001	1000	8
010	1010	1010	1001	9
011	1011	1011	1010	10
100	1100	1100	1011	11
101	1101	1101	1100	12
110	1110	1110	1101	13
111	1111	1111	1110	14

The top row describes the steps of the operations from left to right (indicated by =>) starting from a sequence of 0s and 1s, i.e. some data, and ending with a number, the Gödel number of that data item. The second row describes the steps in the opposite direction (indicated by <=). This way, we have found a way to code any sequence of 0s

[4] For example, a binary number 1011 can be read as $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11$.

and 1s as a natural number. Moreover, this code or mapping as well as the inverse operation, i.e. the decoding of a number to yield the corresponding sequence of 0s and 1s can easily be calculated.

4. Why do we need Gödel numbers?

There are many other possibilities to map data to natural numbers and vice versa. This one is just an example. Its advantage is that it is simple and an actual implementation of it would run quite fast.

However, we do not normally need to actually compute Gödel numbers from data or the other way around. The idea of Gödel numbers is used just to show that it is possible to map data to natural numbers in a computable one-to-one way, so statements we prove for natural numbers, sets of natural numbers and functions or predicates on them apply to arbitrary data as well.

For example, to see what is computable by algorithm, we only need a formalism operating on natural numbers, like the formalism of "recursive functions" mentioned above. This simplifies proofs: we do not need extensive case distinctions for all possible types of data. Once we know it is possible, or impossible, to do certain operations on natural numbers, we can extend that insight to other kinds of data.

For example, I have sketched a proof in previous installments that shows that the set of all computable predicates on natural numbers cannot be algorithmically enumerated. Any such enumeration is incomplete. The idea of Gödelization allows us to extend this proof to sets of arbitrary data types for which a Gödelization can be constructed.

That the set of all computable predicates of natural numbers cannot be algorithmically enumerated means that the concept of truth cannot be formalized for statements on natural numbers, in the sense that there are always more true statements—even true, computable statements, i.e., statements derivable by some algorithm—than can be computed by any single algorithm or derived within any single formal theory.

Gödelization allows us to generalize this to all kinds of data. We could also say that an algorithmic generally-intelligent or even super-intelligent AI (or "AGI"), i.e. a system that can solve every solvable problem, is impossible in principle, no matter how much knowledge, hardware, computing center size, and electrical energy (or cauldron size, grimoires and wood) you throw at the task of building such a thing.

REFERENCES

(Dean, 2022). Dean, W. "Recursive Functions." In E.N. Zalta and U. Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*. Fall Edition. Available online at URL = <https://plato.stanford.edu/archives/fall2022/entries/recursive-functions/>.

(Revoy, 2025). Revoy, D. "The Amphora of Great Intelligence." *Pepper & Carrot.* 10 September, Available online at URL = <https://www.peppercarrot.com/en/miniFantasyTheater/018.html>.