Inconsistency of the ZFC axiomatic theory and the axiom of infinity.

By: Silvano Mattioli

Abstract In this paper we want to demonstrate the inconsistency of the ZFC axioms as expressed. Starting from the definition of "non-computable number" and using the mapping of the Peano curve we arrive at the absurdity of generating a non-computable number starting from an algorithm.

This conclusion is evidently in contrast with the definition of a non-computable number and therefore creates the paradox that causes the castle of ZFC axioms to collapse.

1. Detail of Paradox

The Peano curve is a function that map the Cantor Domain into a square:

$$c : [0,1] \rightarrow [0,1] \times [0,1]$$

There are various definitions of this curve in the literature, for example the Hilbert definition.

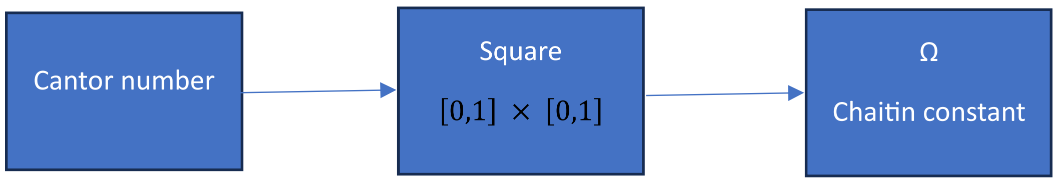
Let's consider the only known non-computable, the number Ω (Chaitin constant, is a number in 0,1 defined starting from the probability problem of stopping the Tuning Machine), for which we know that there is no algorithm capable of calculating it by hypothesis.

It is demonstrated that the Peano curve maps the numbers of the Cantor set into the square [0.1] × [0.1] via a computable function.

Consider the point P (Ω, Ω) that is into the square $[0,1] \times [0,1]$. P it is mapped from a cantor number with an algorithm.

There is an algoritm that map a computable number into a non computable number !

That is impossible for definition of non computable numbers: we have build a non computable number from a computable number with an algoritm.



2. This is an example of algortim of Hilbert curve

From wikipedia:

https://it.wikipedia.org/wiki/Curva_di_Hilbert

```
//Converte (x,y) a d
int xy2d (int n, int x, int y) {
    int rx, ry, s, d=0;
    for (s = n/2; s > 0; s /= 2) {
        rx = (x & s) > 0;
        ry = (y & s) > 0;
        d += s * s * ((3 * rx) ^ ry);
        rot(s, &x, &y, rx, ry);
    }
    return d;
}

//converte d a (x,y)
void d2xy(int n, int d, int *x, int *y) {
    int rx, ry, s, t=d;
    *x = *y = 0;
    for (s = 1; s < n; s = 2) {
        rx = 1 & (t/2);
        ry = 1 & (t ^ rx);
        rot(s, x, y, rx, ry);
        *x += s * rx;
        *y += s * ry;
        t /= 4;
    }
}

//ruotare/capovolgere un quadrante correttamente
void rot(int n, int *x, int *y, int rx, int ry) {
    int t;
    if (ry == 0) {
        if (rx == 1) {
            *x = n-1 - *x;
            *y = n-1 - *y;
        }
        t  = *x;
        *x = *y;
        *y = t;
    }
}
```