"Quaderni di Ricerca in Didattica (Mathematics)", n. 25, 2015
G.R.I.M. (Department of Mathematics, University of Palermo, Italy)

On the impossibility to add and multiply

G. Gerla[1] & F. S.Tortoriello[2]

Abstract: In this article we have adopted modern theoretical computer science as our "higher standpoint", in the hope that this may contribute constructively to a review of the way mathematics is taught in schools. Since there is nothing more elementary than addition and multiplication, the article focuses on these two operations.

1. Introduction

Many mathematicians are familiar with the expression "Elementary mathematics from a higher standpoint" since it denotes a study unit that is common to the majority of mathematics degree courses. The beauty and significance of this expression lie in how it highlights the way elementary notions and problems may be revisited in the light of more recent and complex theories. As is well known, the concept was first put forward in Germany by F. Klein[3], spreading to Italy in the early 1900s, thanks to associations of mathematicians like Mathesis and the Italian Mathematical Union. The purpose of the course is not to render simple things more difficult, but to achieve a deeper understanding of seemingly simple concepts.

2. Finite memory machines: addition and multiplication

Since we have to examine the addition and multiplication algorithms, we feel that it is necessary to say something about the concept of algorithm. Since an algorithm consists of a series of actions that can be performed "mechanically", the concept may be defined as a "machine that performs calculations". Various mathematical models exist for such a machine, and these form the basis of a fundamental part of theoretical computer science: the theory of computation. These models may be divided into two main types: the finite memory machine, represented by the concept of finite automation, and infinite memory machines such as the Turing machine or the register machine (See Minsky's work on Finite and Infinite Machines).

We start by focusing our attention on finite automata, which represent an accurate model for a wide variety of machines, ranging from food distributors and elevators, through to the current computers. Before arriving at a formal definition, let's examine the behavior of a simple machine that many of us use on a day-to-day basis.

---

[1]Department of Mathematics, University of Salerno, gerla@unisa.it
[2] Department of Mathematics, University of Salerno, fstortoriello@unisa.it
[3] Klein, F .: Elementarmathematik vom höheren Standpunkt aus. 2 Bände. Teil 1: Arithmetik, Algebra, Analysis, Teil 2: Geometrie. 2. Aufl. Ausgearbeitet von E. Hellinger. Leipzig: Teubner, *1911+1914*

Elevators with integrated memory systems. Our observations are based on a coin-operated (ridurre spazioe) elevator. Upon entering the elevator car we can provide it with various types of stimuli (inputs), to which it reacts by providing various types of responses (outputs). For example, we may insert a coin, press the alarm button or request the elevator to take us to a specific floor. Assuming that, in addition to the ground floor, there are 7 other levels, these actions may be identified as M, A, 0, 1, 2, 3, 4, 5, 6, 7, respectively. The elevator may respond in a variety of ways: sound the alarm, go to a floor, or even do nothing, i.e. remain stationary. We use A, 0, 1, 2, 3, 4, 5, 6, 7, F to identify these responses. At this point, if we press the A button, while standing inside the elevator car, the elevator responds A, i.e. the alarm is activated. If we insert the requested coin, the elevator does not seem to do anything. The input-output situation may be fully represented by the concept of function. An entirely different phenomenon occurs if we push a floor button. In fact, there are two possible reactions: the elevator may move to the selected floor, or it may remain stationary. Obviously this depends on whether the appropriate coin has been inserted or not. This implies that the elevator must have a basic memory system that is capable of assuming two different states, already paid (denoted by P) and not paid (denoted by N).
Hence:

- at any given instant the machine is either in the coin already received state , or the coin not received yet state,
- the response to an input depends not only on the input that the machine receives but also on its state when it receives it;
- inputs determine not only outputs but also the transition from one state to another.

Therefore, if S = {P, N}, a complete description of the behavior of the coin-operated elevator is given by two functions $O : X \times S \to Y$ and $C : X \times S \to S$ defined as follows, where n is a variable between 0 and 7:

$$\begin{array}{lll} O(n,P) = \underline{n} & ; \quad O(n,N) = F & ; \quad O(A,P) = \underline{A} \\ O(A,N) = \underline{A} & ; \quad O(M,P) = F & ; \quad O(M,N) = F \\ C(n,P) = N & ; \quad C(n,N) = N & ; \quad C(A,P) = P \\ C(A,N) = N & ; \quad C(M,P) = P & ; \quad C(M,N) = P. \end{array}$$

The function O indicates which output is generated by the machine when it receives an input $x \in X$ while in the state $s \in S$. The function C determines the new state assumed by the machine. For example the equations O(3,P) = 3 and C(3,P) = N indicate that the machine does two things when it receives the input 3 while in the already paid state: it goes to the third floor and assumes the not paid yet state.

On the basis of this example it is possible to propose the following general definition for a finite memory machine.

Definition 1.1. A finite automaton consists of:

- three finite sets X, Y, S , identified as inputs alphabet, outputs alphabet and set of states, respectively,
- a function $C : X \times S \to S$ known as the state change or transition function
- a function $O : X \times S \to Y$ known as the output function.

In some cases a specific state s_0 is also determined; this is known as the initial state . The set of states S is also known as the automaton memory, and the hypothesis of finiteness of S confirms our intention to carry out a study on finite memory machines. As has already been stated, the two

functions C and O represent the behavior of the automaton in the sense that:
- if the automaton receives the input x while it is in the state s
- it then 1. assumes the state s' = C(x,s)
2. and generates the output y = O(x,s).

A precise description of the behavior of an automaton is given by the following definition.

Definition 1.2. Given an automaton, for each sequence $x_1, \ldots, x_n$ Input is a sequence $s_1,\ldots,s_n$ and a sequence of states $y_1,..,y_n$ output to the next mode.

$$\begin{array}{lcl} s_1 = C(x_1,s_o) & ; & y_1 = O(x_1,s_o) \\ s_2 = C(x_2,s_1) & ; & y_2 = O(x_2,s_1) \\ \ldots & ; & \ldots \\ s_n = C(x_n,s_{n-1}) & ; & y_n = O(x_n,s_{n-1}). \end{array}$$

It is important to note that the definition of a finite automaton includes all the types of digital (i.e. non-continuous mode) machines that we encounter during our everyday lives. In particular, a computer may be defined as a finite automaton whose state at any given time is represented by how its memory registers are configured at that instant. It goes without saying that, irrespective of the enormous number of states that a computer can assume, this number remains finite.

3. **Adding and multiplying with a finite automaton**

Having arrived at a mathematical definition of a finite memory machine, we wish to demonstrate that the addition operation may be performed by an automaton. Before proceeding, it is necessary to clarify that when we say "may be performed by an automaton" we refer to a sequential operation where the digits of the two numbers to be multiplied, reading from right to left, are received at the input, after which the output digits are printed out. This operation remains the same, regardless of the number of digits in these numbers. This means that X = {0, ..., 9} × {0, ..., 9} is assumed as the set of the inputs, and Y = {0, ..., 9} as the set of the outputs. This automaton may be represented by the image in the figure 1, where s_i corresponds to the state it assumes at a given instant.

Assuming that the two numbers to be entered consist of the same number of digits does not

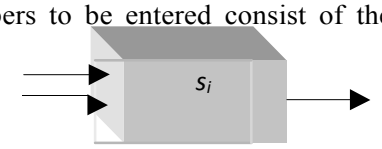


Figure 1.

represent a restriction, since it is always possible to add as many zeroes as necessary at the beginning of the number. It should also be noted that, due to the carry-over phenomenon, the sum of two integers may have an additional digit with respect to the two initial numbers; this means that, after the automaton has completed reading the two numbers, it is necessary to perform an extra step. This problem is overcome easily by rewriting the two numbers with a leading 0. Then again, this is what we normally do with the usual addition algorithm. If we have to sum 972 and 48, we arrange the two numbers in columns and proceed as follows:

```
 972+
  48 =
1020
```

however, we interpret this procedure as follows:

```
0972+
0048 =
1020
```

Theorem 3.1. A finite automaton exists that is able to sum two natural numbers (regardless of the number of digits they contain).

Proof.: Let's start with a look at how we perform additions in base ten. For example, returning to the sum of the two numbers 0972 and 0048. We read the digits (2,8), (7,4), (9,0) and (0,0), one pair at a time, as the inputs, and write 0, 2, 0, 1, as the output, where the latter digits depend both on the input readings, and the carry-over, which we are required to remember for each step of the calculation. The situation is similar to that of the elevator insofar as the system requires very little memory capacity: all we need to remember is "carry over" or "don't carry over". Hence it appears natural to consider an automaton where

- the sets X and Y represent the input and output sets, respectively
- the state S = {R, NR} represents the set of the states
- NR represents the initial state
- the functions O and C are defined as follows:

O(0, 0, NR) = 0, O(0, 1, NR) = 1, … O(0, 9, NR) = 9,
O(0, 0, R) = 1, O(0, 1, R) = 2, … O(0, 9, R) = 0,
O(1, 0, NR) = 1, O(1, 1, NR) = 2, … O(1, 9, NR) = 0,
O(1, 0, R) = 2, O(1, 1, R) = 3, … O(1, 9, R) = 1,
. . . .
O(9, 0, NR) = 9, O(9, 1, NR) = 0, … O(9, 9, NR) = 8,
O(9, 0, R) = 0, O(9 ,1, R) = 1, … O(9, 9, R) = 1.

C(0, 0, NR) = NR, C(0, 1, NR) = NR, … C(0, 9, NR) = NR,
C(0, 0, R) = NR, C(0, 1, R) = NR, … C(0, 9, R) = R,
C(1, 0, NR) = NR, C(1, 1, NR) = NR, … C(1, 9, NR) = R,
C(1, 0, R) = NR, C(1, 1, R) = NR, … C(1, 9, R) = R,
. . . .
C(9, 0, NR) = NR, C(9, 1, NR) = R, … C(9, 9, NR) = R,
C(9, 0, R) = R, C(9, 1, R) = R, … C(9, 9, R) = R.

It is obvious that this automaton is able to sum two numbers regardless of the number of digits they contain.

The situation changes completely when we switch to multiplication, which presents memory problems that are insurmountable for a finite automaton. This is no secret, and everyone will be aware that multiplication involves dealing with problems that simply do not exist when performing

additions. Below we compare the "pen and paper" method that schoolchildren are taught to use when adding two multi-digit numbers, with the equivalent procedure used for multiplication.

972 +	972 ×
48 =	48 =
1030	7776
	38880
	46656

In the first case we are able to carry out the operation mentally, reading the addends one at a time and calculating the result. Our memory is perfectly capable of remembering the two states "carry over" and "do not carry over." The demands placed on the memory in order to proceed with the calculations are largely independent of the length of the numbers involved. In the second case, if both numbers have multiple digits, we are no longer able to remember the results of the intermediate calculations and feel the need to write them down on a sheet of paper. Moreover, even supposing we had the ability to remember two long numbers, if we were to substitute 48 with 2348, at this point it would be necessary to remember three numbers. In general, we need to use the same number of rows as there are digits in the smallest number involved in the calculation. Hence, the larger the two numbers to be multiplied, the greater the space (or memory) required to perform the calculation.

We now return our attention to finite automata and their ability to perform multiplications. In this case it should be noted that the number of digits in the output may be twice that of the inputs. For example the number 99 × 99 = 99 * 99 = 9801 has 4 digits, the number 999 × 999 = 998 001 has 6 digits and so on. In such cases it is sufficient to double the number of digits in the multiplicands by adding the appropriate number of zeroes. For example, in order to calculate the product of 972 and 48, we multiply 000972 by 000048, obtaining the result 46656.

Theorem 3.2. No finite automata exist that are capable of performing multiplications.

Proof.: If such an automaton were to exist then it would have to be able to calculate products in the order of 10m×10m. Obviously, in this case, as the automaton reads the inputs (0,0) from right to left, it must be able to store the number of pairs that it has read in the memory. This means that it must assume the state s(1) after reading the first pair, the state s(2) after reading the second pair, ... the state s(m) after reading pair m. These states must all be different from each other, if for example s(7) were the same as s(3) then 107× 107 would be the same as 103× 103. However, this means that, in order to perform multiplication operations where there is no limit to the number of digits, an automaton would have to have an infinite number of states.

3. Isolated automata and the concept of infinite carry-over

Another limitation of automata becomes apparent as the consequence of a rather surprising phenomenon. If we provide an automaton with the same input repeatedly, after a certain number of steps the sequence of the states, and hence of the outputs, enters a loop that is repeated indefinitely.

Theorem 4.1. If we supply an automaton with inputs that are always exactly the same, sooner or

later the states and inputs sequence will enter a finite loop. More specifically, if n is the number of states, and p is the length of the cycle, then the loop will start after a number h ≤ n-p steps.

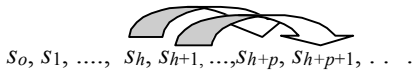
Proof.: When both the initial state s_0 and the input x are constant, the states will evolve as follows:

$$s_1 = C(\underline{x},s_0),\ \ldots,\ s_i = C(\underline{x},s_{i-1}),\ \ \ldots$$

After n +1 steps, the machine must have assumed a state s_j, which it had already assumed before otherwise more than n different states would exist. Therefore $s_j=s_h$ with $h < j \leq n+1$. Assuming p = j-h,

$$s_h = s_j = s_{h+p}$$
$$s_{h+1} = C(\underline{x},s_h) = C(\underline{x},s_j) = s_{j+1} = s_{h+p+1}$$
$$s_{h+2} = C(\underline{x},s_{h+1}) = C(\underline{x},s_{j+1}) = s_{j+2} = s_{h+p+2}$$
$$\ldots$$
$$s_{h+p} = C(\underline{x},s_{h+p-1}) = C(\underline{x},s_{j+p-1}) = s_{j+p} = s_{h+2p}.$$

Therefore, since $s_h = s_j$, it follows that the sequence $s_h\ s_{h+1}\ \ldots\ s_{h+p}$ is equal to the sequence $s_{h+p}\ s_{h+p+1}\ \ldots\ s_{h+2p}$. Also, since $s_{h+p} = s_{h+2p}$ it follows that the sequence $s_{h+p}\ \ldots s_{h+2p}$ is equal to the sequence $s_{h+2p}\ldots s_{h+3p}$. Finally, if the machine receives a constant input, after having assumed a finite sequence of states $s_o, s_1, \ldots, s_h$, starting from point s_h it enters a loop $s_h, s_{h+1}\ldots, s_j$ of length p which is repeated indefinitely. The following figure illustrates what happens



As far as the outputs are concerned, it is sufficient to observe that, given the fact that the inputs always coincide with $\underline{x}$, for each index i $y_i = O(\underline{x},s_i)$. Hence, since the states enter a finite loop, the outputs are also repeated cyclically.

It is immediately apparent from this theorem that there is at least one operation that a finite automaton is not capable of performing, and that, surprisingly, there exists a link between the concept of a finite automaton and the decimal representation of a rational number.

Corollary 4.2. An automaton capable of printing all the digits of $\sqrt{2}$indefinitely does not exist. In more general terms, a finite automaton capable of printing all of the digits of a real number can only exist if such a number is rational.

Proof: As in the case of addition and multiplication, it is necessary to define our terms a little before proceeding. Imagine that the automaton is equipped with an ENTER input-key, and that each time this button is pressed the input corresponds to an additional digit of $\sqrt{2}$. Equivalently, it is necessary to imagine the automaton operating in stand-alone mode, i.e. without any inputs, endlessly printing the figures that follow the decimal point of the number in question. At this point we find ourselves in the situation described by Theorem 4.1, where the automaton must necessarily go into a loop and print figures that are repeated cyclically. Since $\sqrt{2}$is not a rational number, this is impossible

It is also interesting to reformulate Theorem 4.1 in terms of isolated systems.

Definition 4.3. An automaton with a single possible input may be defined as a finite isolated system.

Naturally, it is possible to replace the single input hypothesis with the no input hypothesis, since we are able to image a situation where the constant input is supplied by some automatic mechanism and not by outside intervention. A finite isolated system is identified by two functions C : S → S and O: S → Y. The study of an isolated system coincides with the study of the behavior of any automaton supplied with a constant input x. We can interpret the behavior of an isolated system as:
- "Departure", which corresponds to the first time the system receives the input x,
and
- "evolution", which corresponds to the subsequent introduction of the input x.
Thus it is possible to prove that, if an automaton is "isolated" from the outside world, it will enter a finite loop sooner or later.

Theorem 4.4. If we consider an isolated system, then we can see that the history of the system will sooner or later enter into a cycle that repeats itself endlessly.

Proof. The reasoning is the same as applied in proposition 4.1. Let us consider the way in which the states evolve from a given moment onwards, s_0, s_1, ... It follows from the hypothesis of "deterministic behaviour" that the state s_{i+1} is completely determined by the previous state according to a given law, and therefore that there is a function f such that $s_{i+1} = f(s_i)$. The hypothesis of finiteness means that the machine, after assuming a number of states, must necessarily assume a state that it has already assumed previously. Therefore there exists a state s_j such that $s_j = s_h$ when $h \leq j$. However, assuming $p = j-h$,

$$s_{h+1} = f(s_h) = f(s_j) = s_{p+h+1}$$
$$s_{h+2} = f(s_{h+1}) = f(s_{p+h+1}) = s_{p+h+2}$$
$$\ldots$$
$$s_{h+p} = f(s_{h+p-1}) = f(s_{p+h+p-1}) = s_{2p+h}$$
$$\ldots$$

This theorem may also be applied to the entire universe in which we live, provided we accept the finiteness of the number of states that can may be assumed and the deterministic nature of the system, i.e. that the subsequent state to any given state is determined uniquely[4].

[4] The concept of "infinite carry over" has occupied the minds of numerous philosophers, for example Nietzsche, in *The Gay Science*, states: *"What, if one day or night, a demon were to steal after you into your loneliest loneliness and say to you: "This life, as you now live it and have lived it, you will have to live once more and innumerable times more; and there will be nothing new in it, but every pain and every joy and every thought and sigh, and every unspeakably small or great thing in your life must return to you—all in the same succession and sequence [...]. The eternal hourglass of existence is turned upside down again and again, and you with it, speck of dust!*

5. **Infinite memory machines: addition and multiplication**

We have seen that the difficulties that arise when multiplying integers are the result of the finite nature of memory, and this applies to both machines and humans. It follows that, if we wish to create a mathematical theory of computability that does not exclude this simple operation, it is necessary to disregard the hypothesis of finiteness. At first glance this may seem like a rather questionable solution, since all real machines have finite memory capacity. Nevertheless, such a concept is not entirely without justification. For example, suppose a teacher asks a student to multiply 325467334345 by 57324256453; in all likelihood the pupil will be unable to complete the necessary calculations since they would "not fit" on the sheet of paper provided. In this case, rather than conclude that the student is unable to perform the multiplication, the teacher would provide him/her with a couple more sheets and advise him/her to write smaller. In other words, the student would be provided with additional memory. Accepting a mathematical model of a machine with infinite memory capacity means being prepared to spend money on additional memory each time the machine calculation process requires it. On the other hand, "behaving as if the memory were infinite" falls into that most fundamental of mathematical concepts: abstraction. If we are capable of pretending that there is no such thing as a line of infinite length or a zero-dimensional point, then we should have no problem accepting the concept of a machine with (potentially) infinite memory capacity.

The most famous mathematical model where such a choice has been made is the Turing Machine, which is based on the idea of an infinite tape divided into a great number of cells. It is possible to write letters from a finite alphabet to each cell in such a way that, even though the number of letters is not infinite, it is not subject to any a priori limitations. However, other mathematical models for a "machine that performs calculations" exist that come closer to current computers. For example, Register Machines that function by means of programs and use memory systems based on the ability to write finite sequences of ones and zeroes to certain registers, but without imposing any predefined limits on the length of such sequences. It is beyond the scope of this article to provide more precise definitions of a Turing machine, a register machine, or a program for a register machine, therefore, if readers require additional information on such matters, they should consult the literature on computability theory, for example, the aforementioned book by M. Minski, or the two volumes published by G. Gerla. For the purposes of this article, we refer to the concept of a program as defined in commercially available programming language and assume that the reader understands the idea of a computer executing a program.

The following definition illustrates the basic concepts of computability theory, where it shall be recalled that the *characteristic function* of a set $X \subseteq N^p$ is the function $c_X : N^p \to \{0,1\}$ such that $c_X(x) = 1$ if $x \in X$ and $c_X(x) = 0$ otherwise.

Definizione 5.1. A function $f : N^p \to N$ may be said to be *computable* if a program capable of computing it exists. A subset X of N^p may be said to be *decidable*, if its characteristic function $c_X : N \to \{0.1\}$ is computable, X is *effectively enumerable* if the target set of a total computable function $h : N \to N^p$.

It is important to note the difference between decidable and effectively enumerable. In fact, X may be said to be decidable if a program exists that, after receiving and processing an input x, produces

the response 1 if $x \in X$ or 0 if $x \notin X$. Briefly, X may be said to be decidable if a program exists that can tell whether an item x belongs to X *or not,* whereas, X may be said to be effectively enumerable if a program exists that is capable of generating all the elements of X, i.e. by "printing" them one after the other in the sequence $h(1)$, $h(2)$, ... Since the set of possible programs is enumerable, for reasons of cardinality it is evident that the majority of functions are not computable and that most of the subsets of N are neither decidable or effectively enumerable. The following theorem is a little more difficult to prove, which means that we are obliged to ask the reader to accept the statement at face value.

Theorem 5.2. There exists a set S that it is effectively enumerable but not decidable.

A typical example of this is the set of (code numbers of) the theorems of arithmetic used in first-order logic. In fact, while these theorems may be produced one after another, it is not decidable whether an arithmetical assertion may be considered a theorem or not.

Returning to addition and multiplication, it is no surprise that infinite memory machines are able to perform these operations on integers.

Proposition 5.3. A program exists that is capable of adding two integers. A program exists that is capable of multiplying two integers.

Proof: A rigorous proof of this proposition would require a set of definitions that the available space does not allow us to reproduce in this document. Therefore, here we limit ourselves to indicating two programs that are capable of carrying out these operations, in the hope that they are readable. The following program is able to perform additions using the *'next'* function:

input x ; input y
c : = 0 and s: =x
2. c := c+1
if $c<y$ put s :=s+1 go to 2
output s

Of course it must be assumed that the language concerned is able to calculate the truth value of $c<y$ and the successor of a number. If we denote the addition function defined in this way as $s(x,y)$, we can write the following program, which is able to perform a multiplication as a series of additions:

input x ; input y
c : = 0 and p: =x
2. c := c+1
if $c<y$ put p := $s(p,x)$ go to 2
output p

Note that in these programs, the variables x, y, c, s, p represent memory registers where it is possible to store the value of any integer, without any limits on its size.

We now move on to the field of real numbers, which we identify by their corresponding decimal expansions, taking care not to use the period 9 in order to avoid ambiguity. Hence, an algorithm used to calculate the sum $s = x + y$ of two numbers x and y must be able to calculate the integer part of s and, for any integer n, the nth decimal place of s. Unfortunately, B H Mayoh has proved the following theorem, which demonstrates the impossibility of such an undertaking.

Theorem 5.4. No infinite memory machine is able to multiply by 3 in the set of decimal expansions of real numbers. It follows that none of these machines is able to perform the additions.

Proof: In accordance with theorem 1, we shall consider a set S that is effectively enumerable but not decidable, and assume that $h : N \to N$ is a computable function capable of enumerating S. Furthermore, we use the expression $x = 0{,}x_1...x_i...$ to denote the real number defined by assuming

$x_i = 3$ if $h(i) \neq n$ (i.e. if $n \notin S$)

$x_i = 0$ if $h(i) = n$ (i.e. if $n \in S$).

The value will be a number along the lines of 0.333 ... where the series of 3s will be interrupted by 0 depending on whether n belongs to the set S or not. We affirm that we are unable to calculate the product $3 \cdot x$, or even determine which is the integer part of the product. It is immediately apparent that

- if $n \in S$, then the integer part of $3 \cdot x$ is 0
- if $n \notin S$ then the integer part of $3 \cdot x$ is 1.

Therefore, if we were able to determine the integer part of $3 \cdot x$ we would be also able to decide if n belongs to S or not, in contrast with the undecidability hypothesis of S.

From the impossibility of multiplying by 3, it follows that it is impossible to perform the sum, since $3 \cdot x = (x+x)+x$.

This demonstration is based on concepts that belong to computability theory. It is possible to offer simplified "demonstrations", however not everyone would be willing to consider them satisfactory (see below):

Demonstration illustrated by the use of a bet. Imagine we address a student, Carlo, as follows:

> *we bet that, if I provide you with the decimal expansion of a number x, you will not even able to tell me which is the integer part of the decimal expansion of* $3 \cdot x$.

Once the student accepts the bet it is necessary to clarify the rules of the game: "I undertake to communicate the integer part of x followed by all its decimal places, one after another." Sooner or later, Carlo must start to provide a response, indicating which is the integer part of $3x$, followed by its decimal places. Suppose I then say to Carlo "*the integer part of the number x is 0, the first digit is 3, the second is 3, ...*and so on."

Sooner or later Carlo will be obliged to indicate which is the integer part of $3x$. Now:

- if, after the m^{th} digit, Carlo states that the integer part of $3 \cdot x$ is 0, I will reply that he is wrong because all the subsequent digits of my number are equal to 3, and therefore, since $x = 1/3$, the number $3x$ must be the same
- however, if Carlo informs me that the integer part is 1, I will, again, reply that he is wrong, because all the digits after the m^{th} decimal place are equal to 0, and therefore $x<1/3$ and $3x<1$.

Naturally, not everyone would accept this is as a viable demonstration and in fact it has more than a little of the trick question about it; it would have been fairer to fix the value of the number before starting, by writing down it on a piece of paper and handing it over to a notary, for example. But, in reality, this wouldn't be a problem. I could write down the value of the real number on the piece of paper, and even allow Carlo to read it before starting the experiment. All I need to write is "x corresponds to the number $0{,}x_1x_2...x_i...$ whose integer part is 0, and

$x_i = 3$ if, at the i^{th} instant, Carlo has not yet provided me with the first decimal place or has provided the answer 0
$x_i = 0$ if, at the i^{th} instant, Carlo has already provided me with the answer 1.

Despite this explicit definition of x, there is no chance that Carlo will be able to come up with the correct value of the integer part of $3x$.

Intuitionistic demonstration. At this point, the proposed demonstration may seem a little "unmathematical" and more like the sort of trick that logicians use to prove unlikely concepts. To counteract this, we propose a demonstration that makes use of intuitionistic mathematical techniques. For example, let us consider the question of whether the sequence of numbers 0123456789 occurs in the decimal expansion of π or not. We are currently unable to provide an answer to this question. Let us use $A(i)$ to denote the statement: " i is the first integer such that the sequence 012...9 occurs in the first i decimal places of π" and consider the real number $x = 0,x_1...X_i...$ such that:

$x_i = 0$ if $A(i)$ is true
$x_i = 3$ otherwise;

Since we already have an algorithm for the decimal expansion of π, we have defined x using a mathematical algorithm explicitly designed to generate x. Two cases are now possible:

1^{st} *case.* - $A(i)$ is always false because the sequence 0123456789 never occurs and hence $3 \cdot x = 1$
2^{nd} *case.* - $A(i)$ is true for a number i and hence $3 \cdot x < 1$

Therefore, the integer part of $3 \cdot x$ will be equal to 1 in the first case and 0 in all other cases. Since we are not currently able to decide whether the first or second case is true, we are not able to determine the integer part of $3 \cdot x$.

6. An argument for a definition based on Cauchy sequences

Teachers have always found it difficult to provide a rigorous definition of real numbers. The sections method is often used, but this has the disadvantages of being boring and unintuitive. It requires considerable effort on the part of mathematicians to think of $0.\underline{3}$ or π as sections, or that it is possible to obtain the section $0.\underline{3} + \pi$ from such sections. Apparently, if you have the mind-set of an engineer, a real number is identified by its decimal representation. But in this case does it mean that π can be represented in both base 2 and base 10? What does it mean when we affirm that number 1/3 is equal to 0.333..? In our opinion, a better method would be to construct the range of real numbers as a quotient of the Cauchy equi-convergence modulus rational sequences ring. When using this approach, a real number is defined as a complete class of Cauchy sequences and its decimal representation is merely the selection of a suitable Cauchy sequence (power series in base 10) from this class. Our preference is based on the fact that this way of looking at the real numbers is the one that comes closest to the way that mathematicians treat them. An additional motive is provided by the following proposition, which is in direction opposition to that of Theorem 5.4.

Theorem 6.1. The sum operation can be effectively performed effectively in the set of Cauchy sequences.

Dem. If I have two programs for calculating the values of the Cauchy rational number sequences $(r_n)_{n \in N}$ and $(q_n)_{n \in N}$, then it is possible to paste these programs together, add a simple operation for

adding rational numbers and obtain a program for calculating the values of the sum $(r_n)_{n\in N}+(q_n)_{n\in N} = (r_n+q_n)_{n\in N}$.

One could argue that every decimal expansion is also a Cauchy sequence and, therefore, that this proposition, in contrast with the theorem of Mayoh, demonstrates that it is possible to sum two decimal expansions. More precisely, given two computable real numbers $x = x_0,x_1... X_i...$ and $y = y_0,y_1 ... Y_i ...$ we can consider the sequences $(r_n)_{n\,\epsilon\, N}$ and $(q_n)_{n\,\epsilon\, N}$ obtained by assuming $r_n = X_0, X_1... X_n$ and $q_n = Y_0, Y_1y_2...y_n$. We obtain two effectively computable Cauchy sequences of rational numbers, therefore we can state that $(r_n+q_n)_{n\,\epsilon\, N}$ is an effectively computable Cauchy sequence that represents $x+y$. Unfortunately, as already demonstrated by the proof in Theorem 5.4, there is no guarantee that it will be possible to transform the resulting sequence into an effectively computable decimal expansion. In fact, although r_n+q_n is a rational number written in finite decimal form, it does not necessarily correspond to the expansion of $x+y$ truncated to n decimal places.

Finally, we can see that similar considerations may also be applied to the other arithmetical operations, as demonstrated by the following proposition.

Theorem 6.2. It is not possible to perform subtractions, divisions, multiplications or extract roots effectively in the set of decimal expansions of real numbers.

Other fairly strange events also occur. For example, while we have seen that multiplication by 3 is not executable, multiplication by 2 *is*, whereas, in the case of expansion of base 3, the opposite applies. Moreover, the transition from a base p to a base q is only (effectively) possible if q divides a power of p.

References

Gerla G., (1981) E' sempre possibile addizionare due numeri reali?, Comunicazione al Convegno Mathesis "La matematica nell'educazione", Luciani Ed. Roma 171-175.
Gerla G., (2013). Cosa può fare un calcolatore?, Vol. I, Dalle prime "macchine" ai moderni calcolatori, scaricabile da Ilmiolibro, Repubblica-Espresso
Gerla G. (2013), Cosa può fare un calcolatore?, Vol. II, Macchine che apprendono ed altro, scaricabile da Ilmiolibro, Repubblica-Espresso.
Mayoh B. H., (1965) Unsolvable problems in the theory of computable numbers, in: "Formal Systems and Recursive Functions", edited by Crossley-Dummett (North-Holland, Amsterdam.
Minsky, M., (1967), Computation, finite and infinite machines. London: Prentice-Hall International, INC.
Mostowski A. (1957),, On Computable Sequences, Fund. Math. 44 37-51.
Tortoriello F.S. (2015)., Nietzsche e la matematica dionisiaca, Archimede
Toffalori C. (2015), Algoritmi, il Mulino,.
Zappa P. (2011), Euclide e la calcolatrice, Archimede 122-125.