CHAPTER 1

Introduction

We discuss informally the notion of **effective procedure** and of **function computable** by means of an effective procedure. This will lead us to single out the main features of an algorithm/computational model. Despite being informal, these considerations will allow us to derive the existence of non-computable functions for any chosen effective computational model. Later these notions and considerations be formalised by fixing a specific computational model, a sort of idealized computer.

1.1. Algorithm or effective procedure, informally

Effective procedures and **algorithms** are part of our everyday life, despite the fact that we do not always refer to them using these terms.

For instance, at the primary school, we are not only taught that given two numbers their sum exists, but the teacher also provides us with a procedure to compute the sum of two numbers!

In general terms, an *algorithm* can be defined as the description of a sequence of *elementary steps* (where "elementary" means that they can be performed mechanically, without any intelligence, and effectively) which allows one reach some objective. Typically, the aim is transforming some input into a corresponding output, suitably related to the input. This could be transforming ingredients into a cake, although normally we are interested in computational problems.

EXAMPLE 1.1. Some examples are:

(1) given $n \in \mathbb{N}$ establish whether n is prime;

(2) find the n^{th} prime number;

(3) derive a polynomial;

(4) compute the square root $\sqrt{n}$;

(5) least common multiple *lcm* and greatest common divisor *GCD*.

Therefore we can think of an algorithm as a black box.

$$\text{in} \rightarrow \boxed{\textit{blackbox}} \rightarrow \text{out}$$

where the transformation is performed by executing a sequence of "elementary" instructions.

If the computation is *deterministic*, i.e., in each state of the system the instruction to execute and the new state it produces are uniquely determined, then each possible

input will uniquely determine the corresponding output (if any, in fact the procedure might not terminate, in which case we will have no output).

In mathematical terms the algorithm induces a *(partial) function*

$$f : \{\text{possible inputs}\} \to \{\text{possible outputs}\}.$$

We say that f is the *function computed* by the algorithm and that f is effectively computable. We can thus give the following first definition of a computable function (still informal, since it refers to a generic notion of algorithm).

DEFINITION 1.2 (computable function). A function f is computable if *there exists* an algorithm that computes f.

We stress that for f to be computable, it is not important that we know the algorithm that computes f, but rather we just need to know that some algorithm computing f exists.

EXAMPLE 1.3. According to the above definition, we expect the the following functions to be computable.

- GCD (greatest common divisor), e.g., exploiting Euclid's algorithm.
- the function $f : \mathbb{N} \to \mathbb{N}$ defined as

$$f(n) = \begin{cases} 1 & n \text{ is prime} \\ 0 & \text{otherwise} \end{cases}$$

- $g(n) = n$-th prime number
  (this is computable, maybe inefficiently, by generating numbers and testing for primality, until the n-th prime is found)
- $h(n) = n$-th digit of the decimal representation of π.

  In fact, from analysis we know that
  - There are series that converge to π
  - There are techniques to over-estimate the error caused:
    * by truncating a series
    * by rounding in the calculation of the value of the truncated series

What about the function below?

$$g(n) = \begin{cases} 1 & \text{there is a sequence of exactly } n \text{ consecutive 5's in } \pi \\ 0 & \text{otherwise} \end{cases}$$

For example $g(3) = 1$ if and only if $\pi = 3.14 \ldots k555h \ldots$, with $k, h \neq 5$.

It is unclear whether the function is computable. A naive algorithm could be:

- generate the digits of π
- until a sequence of 5's of the desired length is found.

Clearly, if such a sequence exists, it will be eventually found and the answer 1 will be given. However, apparently, at no point the computation, not having found the sequence of n 5's, we can exclude that it might appear later!! Hence, apparently we have no way of answering 0.

REMARK 1.4. Clearly, something of the kind:

- generate all digits in the decimal representation of π;
- if they include a sequence of n consecutive 5's $\rightarrow g(n) = 1$
- otherwise $\rightarrow g(n) = 0$

is *not* an effective procedure.

Note that this doesn't mean that g is not computable, i.e., that an effective procedure cannot be found (e.g. on the basis of some mathematical properties of π), but at the moment this procedure is not known! (at least by me!) In particular, it is conjectured that all finite sequences of digits appear in π, which would imply that g is the constant 1, whence computable.

Consider now a slightly different function $h : \mathbb{N} \rightarrow \mathbb{N}$, defined by

$$g(n) = \begin{cases} 1 & \text{there is a sequence of at least } n \text{ consecutive 5's in } \pi \\ 0 & \text{otherwise} \end{cases}$$

The function seems very similar to the one considered above. However, note that if $\pi = 3.14 \ldots k555h \ldots$, then we deduce, not only that $h(3) = 5$, but also $h(2) = h(1) = h(0) = 1$. More generally, whenever $h(n) = 1$ then $h(m) = 1$ for all $m < n$. This suggests that h is quite "simple" and "regular".

More precisely, consider $K = sup\{n \mid \pi \text{ contains } n \text{ consecutive digits } 5\}$. Then we have 2 possibilities:

(1) K is finite, and thus $h(n) = 1$ if $n \leqslant k$, 0 otherwise

(2) K is infinite, and thus $h(n) = 1$ for all $n \in \mathbb{N}$

This implies that h is computable because it is either a step function or a constant function, that are computed by very simple programs. One could object that we don't know which shape the function has and thus we do not know the program that computes the function. This is true but it doesn't mean that the function is not computable, only that we do not know how to compute it!

Observe that if tried to argue about the computability of function g along a similar pattern we would have failed. In fact, one could think of defining $A = \{n \mid \pi \text{ contains exactly } n \text{ consecutive 5's}\}$. Then we would have

$$g(x) = \begin{cases} 0 & x \in A \\ 1 & x \notin A \end{cases}$$

However the set A is possibly infinite and we do not see a way of providing a finite representation of A which can be included in a program. Hence we cannot deduce anything about the computability of g.

Bringing the argument to the extreme, one could consider the function $G : \mathbb{N} \to \mathbb{N}$ defined by

$$G(x) = \begin{cases} 1 & \text{if God exists} \\ 0 & \text{otherwise} \end{cases}.$$

This is either the constant 0 or the constant 1. Independently of which of the two cases applies, the function is computable.

CHAPTER 2

Algorithms and existence of non-computable functions

2.1. Characteristics of an algorithm

We present a list of features that an algorithm should satisfy in order to capture the intuitive idea of an effective procedure. Roughly, what we will ask is that it is "implementable" on some sort of idealised machine, the chosen computational model. Hence, in turn, we will list some requirements that the computational model should meet to be considered effective.

An **algorithm** is a sequence of instructions with the following characteristics:

a) it is of **finite length**;

b) there exists a **computing agent** able to execute its instructions;

c) the agent has a **memory** available (to store input, intermediate results to be used in subsequent steps and output)

d) the computation consists of discrete steps (it does not rely on analog devices)*

e) the computation is neither nondeterministic nor probabilistic (we model a digital computer)

f) there must be no limit to the size of the input data
(we want to be able to define algorithms that work on any possible input, e.g. sum ... operating on summands of any size);

g) there is no limit to the memory that can be used
This requirement could appear less natural, but having an unbounded memory is essential to avoid the the notion of computability depends on the specific resources which are available. In fact, for many functions the space required for the intermediate results depends on the size of the input.

h) there must exist a finite limit to the number/complexity of the instructions

This is intended to capture the intrinsic finiteness of the computing device (justified by Turing with the limits of the human mind/memory);

E.g. for a computer, the memory that can be accessed with a single instruction must be finite (even if by (g), the available memory is unbounded);

i) computations might

   (a) terminate and return a result after a finite (but unbounded) number of steps (e.g. the square function requires a number of steps proportional to the argument);

   (b) diverge, i.e., continue forever, and return no result.

2.2. Existence of non-computable functions

Later we will focus on a specific computational model, that will allow us to give a completely formal definition of computable function. Here we observe that, quite interestingly, simply on the basis of the assumptions above, we can infer the existence of non computable functions for every "effective" computational model.

We start by recalling some basic notions and introducing useful notation.

- We will consider the set of *natural numbers* $\mathbb{N} = \{0, 1, 2, \dots\}$;
- Given the sets A, B their *cartesian product* is

  $$A \times B = \{(a, b) \mid a \in A \ \wedge \ b \in B\}.$$

  We will write A^n for $A \times A \times A \times \ldots \times A$ n times. More formally $A^1 = A$ and $A^{n+1} = A \times A^n$.
- A (binary) *relation* or *predicate* is $r \subseteq A \times B$.
- A *(partial) function* $f : A \to B$ is a special relation $f \subseteq A \times B$ such that if $(a, b_1), (a, b_2) \in f$ then $b_1 = b_2$. Following the standard convention, we will write $f(a) = b$ instead of $(a, b) \in f$
  - the *domain* of f is $dom(f) = \{a \mid \exists b \in B.\ f(a) = b\}$;
  - we write $f(a) \downarrow$ for $a \in dom(f)$ and $f(a) \uparrow$ for $a \notin dom(f)$;
- Given a set A we indicate with $|A|$ its *cardinality* (intuitively, the number of elements of A, but the notion extends to infinite sets). Given the sets A and B we have
  - $|A| = |B|$ if there exists a bijective function $f : A \to B$;
  - $|A| \leqslant |B|$ if there exists an injective function $f : A \to B$ injective or equivalently[1] a surjective function $g : B \to A$.

  Observe that if $A \subseteq B$ then $|A| \leqslant |B|$ as witnessed by the inclusion, which is an injective function

  $$\begin{array}{rl} i : & A \to B \\ & a \mapsto a \end{array}$$

- We say that A is *countable* or *denumerable* when $|A| \leqslant |\mathbb{N}|$, i.e., there is a surjective function $f : B \to A$. Note that, when this is the case, we can list (enumerate, whence the name) the elements of A as

  $$\begin{array}{cccc} f(0) & f(1) & f(2) & \dots \\ a_0 & a_1 & a_2 & \dots \end{array}$$

[1] Strictly speaking, the equivalence requires the axiom of choice.

- When A, B are countable then $A \times B$ is countable.

  Idea of the proof:

  – Since A and B are countable, we can consider the corresponding enumerations

  $$\begin{array}{cccc} A & a_0 & a_1 & a_2 \\ B & b_0 & b_1 & b_2 \end{array}$$

  and place the elements of $A \times B$ in a sort of matrix

	b_0	b_1	b_2	
a_0	(a_0, b_0)	(a_0, b_1)	(a_0, b_2)	$\dots$
a_1	(a_1, b_0)	(a_1, b_1)	(a_1, b_2)	$\dots$
a_2	(a_2, b_0)	(a_2, b_1)	(a_2, b_2)	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

  in a way that they can be enumerated along the diagonals as follows: $(a_0, b_0), (a_0, b_1), (a_1, b_0), (a_0, b_2), (a_1, b_1), (a_2, b_0), \dots$ (this is referred to as "dove tail" enumeration)

- A countable union of countable sets is countable: if $\{A_i\}_{i \in \mathbb{N}}$ is a collection of countable sets then $\bigcup_{i \in \mathbb{N}} A_i$ is countable.

2.3. Existence of non-computable functions in each computational model

Let us considered some fixed computational model satisfying the assumptions in §2.1. We want to show that there are necessarily functions which are not computable in such a model.

We focus on unary functions over the natural numbers. Let $\mathcal{F} = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$ be the set of all the (partial) unary functions on $\mathbb{N}$.

Let $\mathcal{A}$ be the set of all algorithms in our fixed computational model. Every algorithm $A \in \mathcal{A}$ computes a function $f_A : \mathbb{N} \to \mathbb{N}$ and a function is computable in our model if there exists an algorithm that computes it. Hence the set $F_{\mathcal{A}}$ set of computable functions in the given computational model is

$$\mathcal{F}_{\mathcal{A}} = \{f_A \mid A \in \mathcal{A}\}.$$

Certainly $\mathcal{F}_A \subseteq \mathcal{F}$. But, is the inclusion is strict, i.e., are there non-computable functions?

The answer is yes, essentially for combinatory reasons: algorithms are too few to compute all functions.

In fact, an algorithm $A \in \mathcal{A}$ will be a finite, by assumption (a)), sequence of instructions taken from some instruction set I. Morever, by assumption (h)), I must be finite. Hence

$$\mathcal{A} \subseteq \bigcup_{i \in \mathbb{N}} I^n$$

Since a countable union of finite (hence countable) sets is countable, we have

$$|\mathcal{A}| \leqslant |\bigcup_{n \in \mathbb{N}} I^n| \leqslant |\mathbb{N}|$$

and since the function

$$\begin{aligned} &\mathcal{A} \to F_{\mathcal{A}} \\ &A \mapsto f_A \end{aligned}$$

is surjective by definition, we have that

$$|F_{\mathcal{A}}| \leqslant |\mathcal{A}| \leqslant |\mathbb{N}|$$

On the other hand the set of all functions, $\mathcal{F}$, is not countable. Let $\mathcal{T}$ the subset of $\mathcal{F}$ consisting of the total functions $\mathcal{T} = \{f \mid f \in \mathcal{F} \ \wedge \ dom(f) = \mathbb{N}\}$. We show that

$$|\mathcal{F}| \geqslant |\mathcal{T}| > |\mathbb{N}|.$$

We prove that $|\mathcal{T}| > |\mathbb{N}|$ by contradiction. Let us suppose that $\mathcal{T}$ is countable. Then we can consider an enumeration $f_0, f_1, f_2, \ldots$ of $\mathcal{F}$ and thus a matrix like the following

	f_0	f_1	f_2	
0	$f_0(0)$	$f_1(0)$	$f_2(0)$	$\ldots$
1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$\ldots$
2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

and build a function, that consists of the values on the diagonal, systematically changed:

$$\begin{aligned} &d : \mathbb{N} \to \mathbb{N} \\ &d(n) = f_n(n) + 1 \end{aligned}$$

We can observe that

- d total, by definition;
- $d \neq f_n$ for all $n \in \mathbb{N}$ (since $d(n) = f(n) + 1 \neq f(n)$.

This is absurd, since $f_0, f_1, f_2, \ldots$ is an enumeration of all the total functions.

Summing up:

$$\mathcal{F}_A \subsetneq F \text{ and } |F_A| \leqslant |\mathbb{N}| < |\mathcal{T}| = |\mathcal{F}|$$

therefore $F_A \subsetneq F$, as desired.

REMARK 2.1. Note that the set of non-computable functions is not countable:

$$|\mathcal{F} \backslash \mathcal{F}_{\mathcal{A}}| > |\mathbb{N}|.$$

In fact, $\mathcal{F} = \mathcal{F}_{\mathcal{A}} \cup (\mathcal{F} \backslash \mathcal{F}_{\mathcal{A}})$. Thus, if it were $|\mathcal{F} \backslash \mathcal{F}_{\mathcal{A}}| \leqslant |\mathbb{N}|$, since the union of countable sets is countable, we would have $\|\mathcal{F}| \leqslant |\mathbb{N}$.

We conclude that

(1) no computational model can compute all functions;

(2) the non-computable functions are the majority.