On Turing Completeness, or Why We Are So Many

Ramón Casares

ORCID: 0000-0003-4973-3128

Why are we so many? Or, in other words, Why is our species so successful? The ultimate cause of our success as species is that we, Homo sapiens, are the first and the only Turing complete species. Turing completeness is the capacity of some hardware to compute by software whatever hardware can compute.

To reach the answer, I propose to see evolution and computing from the problem solving point of view. Then, solving more problems is evolutionarily better, computing is for solving problems, and software is much cheaper than hardware, resulting that Turing completeness is evolutionarily disruptive. This conclusion, together with the fact that we are the only Turing complete species, is the reason that explains why we are so many.

Most of our unique cognitive characteristics as humans can be derived from being Turing complete, as for example our complete language and our problem solving creativity.

Keywords: *Turing completeness, problem solving, cognition, evolution, hardware and software.*

This is DOI: 10.6084/m9.figshare.5631922.v2, version 20171226.

Any comments on it to papa@ramoncasares.com are welcome.

On Turing Completeness, or Why We Are So Many

§1 Introduction

§1.1 Why are we so many?

¶1 · Why is our species so successful? I am meaning successful in an empirical way: by counting the number of *Homo sapiens* individuals, or by weighing the mass of all the members of our species. Nevertheless, big numbers on these are also the symptoms of a pest. So, to be clear, let me rephrase the question: Why are we so many?

¶2 · An answer is that we are occupying more and more lands. Another that we can live in high density areas, like cities. These are answers, however, to another question: How are we so many? Of course, a good answer to the why question should also explain how it is achieved, but here in this paper we will focus on the why question.

¶3 · An answer in the right direction is that we are very clever, the cleverest species, so we can use all the resources on Earth for our own benefit. However, this answer explains only as much as the meaning of the word 'clever' is explained. Otherwise the explanation becomes empty or circular.

¶4 · Trying to avoid the emptiness and circularity, there is another answer: we are so many because we are the brainiest species, where braininess is also measured in some empirical way, for example as brain weight divided by body weight. Letting aside any doubts about the truth of the fact that we are the brainiest species, this asks for a new explanation: Why and how does more brain make a cleverer species? It could seem obvious to you, but as far as this question is unanswered, and it still requires explaining the word 'clever', the big brain explanation does not explain anything.

¶5 · Perhaps we need big brains to have a language. This would explain why only the brainiest species has a language, and surely a language is a tool that helps us to survive, but then we should have to address two new questions:

- Why and how does language make us the most successful species?
- How much brain is needed to have a language?

Spoiler: Our answer here to the last question will be that not too much, in fact much less than a chimpanzee brain. So language does not require the biggest brain.

¶6 · Anyway, the importance of properly answering the 'Why are we so successful?' question has prompted many answers, some following the lines mentioned above, and many others from other assumptions and theories. Instead of discussing these many answers here, I will present my own answer as clearly as I can, so that any interested party can judge it by himself.

§1.2 Contents

¶1 · Why is our species so successful? Why are we so many? Because we are Turing complete. Turing completeness is a very interesting computing property: Turing completeness is the capacity of some hardware to do by software whatever hardware does. Then, section §2 is devoted to present computing and completeness as defined by Turing (1936 and 1937). In subsection §2.1, supplemented by annex §7.1, we present the Turing machine, which we use to define what is hardware and what is software. Then, in subsection §2.2, we review the limits of computing, both in theory, where Turing computing is equivalent to recursion, and in practice, where Turing computing is real computing, save for time or tape limitations. In subsection §2.3, we present the universal Turing machine, which is the prototype of Turing complete device, and its complete language. Turing completeness is realizable under different engineering recommendations, as revised

in subsection §2.4. And, in fact, we are Turing complete, as shown in subsection §2.5, because we can imitate any Turing machine, save for time or tape limitations.

¶2 · Our explanation of why being Turing complete is the cause of our success is in section §3. We start in subsection §3.1 arguing that software is much cheaper than hardware, implying that, beyond a breaking point of complexity, Turing complete devices are much more profitable. Then, in subsection §3.2, we revise the computing foundations by Turing (1936) to conclude that computing is for problem solving, and then that a computing device is a resolver. In subsection §3.3, supplemented by annex §7.2, we argue that Turing completeness is evolutionarily disruptive, which is the key statement in the paper, by assuming that evolution is the resolver of the survival problem, and by noting that a Turing complete resolver can solve universally and quickly in cheap software what evolution would solve by a trial and error procedure on generations of material living beings, that is, limitedly and slowly in expensive hardware. In subsection §3.4, we argue that we are the only Turing complete species, because no other species shows a problem solving creativity as ours, by far. And, in subsection §3.5, we explain why *only* us are Turing complete. In summary, we are so successful because only our species is Turing complete and Turing completeness is evolutionarily disruptive.

¶3 · In section §4, we explore some philosophical consequences of being Turing complete. If our species is the only Turing complete species, as we argue, then our species is unique (§4.1). Being Turing complete, we are free to imagine any way to solve our problems in our complete language, so creativity is the mark of Turing complete resolvers (§4.2 and §4.3). Language exceeds reality by adding possibilities, and language exceeds actuality by adding change (§4.4). Will is our verbalized intention (§4.5). Explaining something is expressing its intentionality, so explaining is the sending side of the semantic communication channel, and understanding is the receiving end (§4.6 and §4.7). As both ends of the semantic channel are built in our brains, we experience our will as an inner voice, or an internal dialog, called the stream of consciousness (§4.8). Dualism can be explained if our cognition consists of a Turing complete intentional layer built on top of perception, an observation that is compatible with subjectivism (§4.9). Being Turing complete, we can be instructed to follow whatever rules, and the study of rule systems is the subject of philosophy (§4.10). Culture is the set of learned rules that prevail in a society (§4.11). We even enforce rules on top of the genetic codes and the laws of nature, creating artificial worlds where other species cannot survive, and perhaps not even us if we persist (§4.12).

¶4 · In section §5, we examine the theory presented in this paper. Firstly, in subsection §5.1, we justify why we use computing by Turing: because his theory is realizable. And any further doubts on computationalism are explained away in subsection §5.2, just by acknowledging that *computing is for solving problems*, which is the first law of cognition. To link evolution to problem solving we assume that *solving more problems is evolutionarily better*, resulting a mathematical problem theory summarized in subsection §5.3, where a series resolvers of increasing problem solving power finishes when it reaches a Turing complete brain. The relation between problem solving and brain evolution is confirmed by domestication, as shown in §5.4, though, in subsection §5.5, we note that Turing completeness is not concerned with brain size. Another fundamental cognitive law, *software is much cheaper than hardware*, is discussed in subsection §5.6. All these results together imply that *Turing completeness is evolutionarily disruptive*. And we are so successful because, as stated by the fundamental law of cognition, which is the law of Post presented

in subsection §5.7, *in computing capacity, we are just Turing complete*. Nevertheless, we would not be as successful as we are should we had to compete against other Turing complete resolvers, so the remaining question is: Why are we the only Turing complete species? Why only us? Cooperation, discussed in §5.8, is one of the causes, perhaps the key one, but cooperation is not enough as shown by other cooperative species that are not Turing complete. In subsection §5.9, the relation between cooperation and language leads us to compare the concept of recursion in linguistics with our view here, where we assimilate the recursive languages to the complete languages of Turing complete devices.
¶5 · Finally, in section §6, we conclude summarizing the argument used in this paper to answer the question: Why are we so many?

§2 Turing completeness

§2.1 The Turing machine

¶1 · *Turing completeness* is the capacity of some hardware to compute by software whatever hardware can compute. And therefore, to understand Turing completeness, we have to distinguish hardware from software, and we have to establish the computing limits of hardware. Fortunately, we can follow Turing on both.
¶2 · The classic reference of computing is its founding paper by Turing (1936), where he presents his *a*-machine, now deservedly known as Turing machine, which is his mathematical model for computing devices. A Turing machine is a finite state computing module attached to a potentially infinite tape.
¶3 · The tape is read and write memory, and it is divided in squares, where each square can contain one symbol out of a finite set of symbols, or none. When a square contains no symbol, we say that it is *blank*. For simplicity, we will consider that *blank* is also a symbol, the null symbol. Each square has exactly one neighbor square to the left and one neighbor square to the right, meaning that the tape is infinite and that no computation will abort because of a lack of squares, though the number of non-*blank* squares will always be finite. Another way of seeing it is that the tape is finite but potentially infinite, and that new *blank* squares are created whenever either end of the tape is going to be overstepped.
¶4 · The finite state computing module is a finite-state automaton. As those finite-state automata later investigated by Mealy (1955) and others, the finite state computing module is in each moment in exactly one state out of a finite set of states, and the next state and the output depends on the current state and the input. In the case of the Turing machine computing module, the finite set of states includes an initial state, the input is read from the scanned square of the tape, and the output is a pair composed of a symbol and a movement, where the symbol is written to the scanned square and the movement is *right* or *left* or *halt*, so the next scanned square will be the neighbor one to the right, or the neighbor one to the left, or none halting the computation. The computation is executed in a loop with four stages —read symbol, transition state, write symbol, and move— that is finished whenever the movement is *halt*. So a table defining the next state and the pair of symbol to write and movement to do for each possible current state and read symbol determines completely the Turing machine behavior. We will say that the pair of current state and read symbol is the index, and that the triplet of next state, symbol to write and movement to do is its corresponding instruction. Together, the index with

its corresponding instruction form a 5-tuple. As the number of states and the number of symbols are both finite, the table is finite.

¶5 · Being a finite-state automaton, the finite state computing module of a Turing machine is physically implementable, if the number of states is not astronomic. See that even a small 1 MB memory can store $8 \times 1024 \times 1024$ bits, so it can be in $2^{8\times1024\times1024}$ states, which are a lot. And below, in §2.4, we will see that a tiny amount of memory, much less than 1 MB, is enough to achieve the maximum computing power. Thus, while the tape is unbounded external memory, the states of the finite state computing module are stored in a finite internal memory, where external and internal refer to the finite state computing module.

¶6 · The computation by a Turing machine starts from a well-defined situation. We will assume without loss of generality that in the initial situation the state is the initial state and the scanned square is the leftest non-*blank* square, if any; if all are *blank*, then any one goes, as any one is as good as any other. From that initial situation, the Turing machine follows the instructions given in the table until it reaches a *halt* movement. Some more details and examples are provided in annex §7.1.

¶7 · So the Turing machine transforms the string of symbols that was written on the tape when it started into the string of symbols that was written on the tape when it *halt*ed. From this point of view, each Turing machine implements a function from the set of strings of symbols to the set of strings of symbols. By definition, any function implemented by a Turing machine is a *computable function*. Note that these functions are partial functions, because there is not any guarantee that a Turing machine on an initial string will reach a *halt* movement, and then the function implemented by the Turing machine is not defined for those input strings on which the machine never *halt*s.

¶8 · In Turing's (1936) model of computing, the Turing machine is the computing device, so it is *hardware*, while what is written on the tape is *software*. The Turing machine can also be interpreted as a function, and then what is written on the tape is *data*, because the tape is from where the function arguments, or input data, are read, and to where the function results, or output data, are written. The tape is also used to store intermediate results, or temporal data. Then, data is synonymous with software.

§2.2 Real computing

¶1 · The strings of symbols are equivalent to numbers, because there is a bijective function from the finite strings of symbols drawn from a non-empty set of k symbols to the set of natural numbers $\mathbb{N}$, achieved by letting that each symbol represents a distinct number from one to k. It is called a bijective base-k numeration. For example, if the set of symbols is $\{\mathtt{A},\mathtt{B},\mathtt{C}\}$, by pairing $\mathtt{A} \leftrightarrow 1$, $\mathtt{B} \leftrightarrow 2$, and $\mathtt{C} \leftrightarrow 3$, we can map any number, as for example fifteen, to a string, in this case to string `AAC`, because $15 = 1 \times 3^2 + 1 \times 3^1 + 3 \times 3^0$, and conversely, any string of symbols, for example `ABC`, can be mapped to a number, in this case to eighteen, because $1 \times 3^2 + 2 \times 3^1 + 3 \times 3^0 = 18$. Zero maps to the empty string. Bijective numerations are computable and decidable, and then mathematicians can translate both ways between the computable functions from strings to strings implemented by Turing machines and the functions from natural numbers to natural numbers of arithmetic.

	↔	0
`A`	↔	1
`B`	↔	2
`C`	↔	3
`AA`	↔	4
`AB`	↔	5
`AC`	↔	6
`BA`	↔	7
`BB`	↔	8
`BC`	↔	9
`CA`	↔	10
`CB`	↔	11
`CC`	↔	12
`AAA`	↔	13
`AAB`	↔	14
`AAC`	↔	15

¶2 · Turing (1937), page 153, wrote:

> *The purpose of the present paper is to show that the computable (Turing 1936) functions introduced by the author are identical with the λ-definable (Church 1935) functions of Church and the general recursive (Kleene 1935) functions due to Herbrand and Gödel and developed by Kleene. It is shown that every λ-definable function is computable and that every computable function is general recursive. [. . .] If these results are taken in conjunction with an already available (Kleene 1936) proof that every general recursive function is λ-definable we shall have the required equivalence of computability with λ-definability.*

The diagram can help us to see Turing's (1937) plan to use Kleene's (1936) proof.

$$\begin{array}{ccc} \text{recursive function} & \Rightarrow & \lambda\text{-definable function} \\ \Uparrow & & \Downarrow \\ \text{computable function} & & \end{array}$$

¶3 · Kleene (1936) had already shown the equivalence of λ-definability with recursion "by proving that all recursive functions, in a wide sense of the term recursive, due to Herbrand and Gödel, are λ-definable; and conversely, all λ-definable functions of the type in question are recursive" (page 343).

¶4 · We can express the identity of computable with recursive functions proved by Turing (1937), with the help of Kleene (1936), as a mathematical theorem:

every recursive function is computable

and

every computable function is recursive.

¶5 · This theorem states the limits of Turing machines: the set of all functions implemented by all Turing machines is equal to the set of all recursive functions from natural numbers to natural numbers. As a mathematical theorem, it applies to mathematical objects, and here we are mostly interested in real computing, so we need to see how it applies to real computing devices.

¶6 · Turing's (1936) model of computing separates cleanly the computing capacity from the infinities. A Turing machine has not time limitations, and it is a finite-state automata attached to an infinite tape, which is just external memory with some minimal requirements: read and write one symbol, move one position. This way computing abstracts away the limitations of a device in speed and memory from its computing capacity. In other words, any real computation is a Turing computation, though some Turing computations would be aborted when performed by real computing devices because of a lack of time or a lack of external memory. Real computing is Turing computing; and Turing computing is real computing, save for lack of time or lack of tape. And neither lack of time nor lack of tape refer to the computing capacity of the finite state computing module.

¶7 · In summary, all computing results derived from Turing (1936) can be applied to real computing just by adding the "save for time or tape limitations" provision. In particular, save for time or external memory limitations, any recursive function can be implemented by real hardware. It should be stressed that these limitations refer to available time and

to external memory capacity, and that they do not refer to computing capacity. Therefore, under Turing's model, real computing approaches theoretical computing as much as desired, just by adding more time and more external memory to real computing. So, Turing's computing is real computing in the limit, and the approximation is independent of computing capacity. That is, in computing capacity, Turing's computing and real computing are exactly the same. This is, perhaps, why Turing's model is so successful.

§2.3 The universal Turing machine

¶1 · The universal Turing machine was also defined by Turing (1936). This is the first paragraph of section 6, pages 241–242, titled "The universal computing machine":

> *It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with a tape on the beginning of which is written the S.D [standard description] of some computing machine $\mathcal{M}$, then $\mathcal{U}$ will compute the same sequence as $\mathcal{M}$. In this section I explain in outline the behavior of the machine. The next section is devoted to giving the complete table for $\mathcal{U}$.*

¶2 · This means that a universal Turing machine is a Turing machine that can mimic the behavior of any Turing machine, including the universal ones. To do it, part of the input data entered to the universal Turing machine is a detailed description of the Turing machine to imitate, a description that we will call *program*. Those programs are written on the tape, so they are software, and therefore universal Turing machines are Turing complete, because they have the capacity to compute by software whatever hardware can compute. In fact, the universal Turing machine is the mathematical model for Turing complete devices, and Turing completeness is also known as *universal computing*.

¶3 · Turing complete devices need a language in which to write those programs. We will call the language of a Turing complete device a *complete language*. For example, Turing (1936) used Turing machines' standard descriptions S.D as complete language, but this is not the only possible complete language. Any complete language has to fulfill two requirements: the syntactic requirement and the semantic requirement.

¶4 · The syntactic requirement refers to the set of well-formed expressions, in this case, the set of well-formed programs. The Turing complete device has to imitate any Turing machine, and therefore it is required that we can express any Turing machine in the complete language. The number of Turing machines is equal to the number of natural numbers, as proved by Turing (1936), where the set of natural numbers $\mathbb{N}$ is an infinite enumerable set. This means that any infinite enumerable set can be used as syntax of a complete language, because then we can map each possible Turing machine to a different syntactic object, and then any Turing machine can be expressed in the complete language. Now we can already state the syntactic requirement for completion: the set of syntactic objects has to be infinite enumerable, or bigger. And, dealing only with finite strings of symbols drawn from a finite set, as it is the case of Turing machines, not even bigger. In mathematical terms, $|S_c| = \aleph_0$, where S_c is the set of the syntactic objects of a complete language, and $\aleph_0$ is the smallest infinite cardinal number, which is also the cardinality of the natural numbers, $\aleph_0 = |\mathbb{N}|$.

¶5 · Any infinite set can be the syntax of a complete language, because then we can express any Turing machine in the language. But, in addition, the Turing complete device has to take any syntactic object expressing a Turing machine and effectively imitate it. This is the semantic requirement for completion, because the Turing complete device

has to know the meaning of the syntactic object to perform the imitation. Viewing each Turing machine as the function that it implements, and given the equivalence of the computable functions with the recursive functions proved by Turing, the syntactic requirement grants that any recursive function is expressible in the complete language, and the semantic requirement grants that any such expression effectively results in the calculation of the recursive function by the Turing complete device. Because of this, we will call any semantics needed to fulfill the semantic requirement for completion a *functional semantics*. And because any recursive function can be expressed and calculated in a complete language, we say that a complete language is a *recursive language*.

¶6 · An important mathematical property of any complete language is full reference.

PROOF Gödel (1930) showed that there is an effective bijection $\mathcal{G}$ between the set of finite strings of symbols drawn from any non-empty enumerable set and a proper subset of the natural numbers, now known as Gödel numbers, $\mathbb{G} \subset \mathbb{N}$. So, if Σ is a non-empty finite set of symbols, and Σ^* is the set of finite strings of symbols drawn from Σ, then it exists a Gödel numbering $\mathcal{G} : \mathbb{G} \leftrightarrow \Sigma^*$. It is required that both $\mathcal{G}$ and $\mathcal{G}_\in$, which is the characteristic function of $\mathbb{G}$, $\mathbb{G} = \{x \mid \mathcal{G}_\in(x)\}$, are computable and decidable, so we can effectively go from any string to its corresponding Gödel number and, in the other direction, we can effectively determine whether or not a number is a Gödel number, and if it is, then we can effectively go from that Gödel number to its corresponding string. As the strings of symbols are equivalent to numbers, see §2.2, we have another natural bijection, the bijective numeration $\mathcal{N} : \mathbb{N} \leftrightarrow \Sigma^*$, which is also computable and decidable. We will call string $\mathcal{N}(g)$, when $\mathcal{G}_\in(g)$ is true (meaning that g is a Gödel number), the name of string $\mathcal{G}(g)$. It is always possible to implement Gödel and natural numberings in a complete language, because all computable functions are available in it. Therefore, in any complete language there are means to effectively name any string of symbols, and thus any complete language is full referable. Q.E.D.

§2.4 Real universality

¶1 · A universal Turing machine is a Turing machine, and therefore what we have seen in §2.2 applies: any real Turing complete device computes exactly as a universal Turing machine, save for time or tape limitations, that is, provided that the real Turing complete device has enough time to perform the computation, and that it can access as much external memory as it needs. This means that any recursive function can be programmed into and calculated by a real Turing complete device, save for time or tape limitations.

¶2 · The function is the mathematical concept that abstracts change. Whenever something is transformed into something, we can use a function to describe mathematically the transformation. For that reason, functions are widely used in Physics, and in Science generally. And then we can say that Turing completeness is the computing capacity of some real devices to calculate, save for time or tape limitations, any algorithmic rule for change given to them as data. Here 'algorithmic' is synonymous with 'recursive', with 'computable', and with 'programmable'; please, serve yourself.

¶3 · In §2.1 we left pending a question: How much internal memory is enough? Although it seems that it cannot be answered, enough for what, in fact it can be answered because Turing completeness is the maximum computing capacity. Any computation, which is any computation that any Turing machine can compute, can be computed by a universal Turing machine, and the universal Turing machine is a Turing machine, so it has a finite number of states stored in its finite internal memory. Therefore, the finite internal

memory of a universal Turing machine is enough to compute any computation at all, and the question can then be reformulated: How much internal memory does a universal Turing machine need?

¶4 · Shannon (1956) showed "that a universal Turing machine can be constructed using one tape and having only two internal states." Two is a small number indeed, but as Shannon finished, what is interesting "is to find the minimum possible state-symbol product for a universal Turing machine", because the state-symbol product is the size of the Turing machine table. And, for example, Neary and Woods (2006) found universal Turing machines with the state-symbol products 3×11, 5×7, 6×6, 7×5 and 8×4, for two-symbol imitated Turing machines, *blank* and another symbol, and using coders and decoders that add complexity that is not accounted for in those products, see Dershowitz and Falkovich (2012). These are not the last values, because mathematicians are still investigating, but they show us that Turing machines internal memories do not need to be astronomical, and in fact internal memories can be tiny, even those of universal Turing machines. The conclusions are that the finite state computing modules of Turing machines, including universal Turing machines, are realizable, and that, above some tiny amount, the computing capacity of a device is independent of its internal memory.

¶5 · Universal Turing machines are Turing machines, so they are mathematical objects under Turing's model of computing. As seen above, this means that real hardware can be Turing complete, and in fact some real hardware is Turing complete. Any full-programmable computer is Turing complete because, save for time or external memory limitations, it can be programmed to compute any recursive function. We are sure because we can program it to execute a conditional loop on the 5-tuples defining any Turing machine, so the imitation will be exact, save for time or tape limitations. For example, and to compare ideal mathematical complexity with practical engineering complexity, the Intel 4004 microprocessor used 2 300 transistors, where a microprocessor is a Turing complete finite state computing module in a chip, and the 4004 was in 1971 the first commercially available microprocessor; see Betker, Fernando & Whalen (1997). And while mathematicians were trying to decrease the state-symbol product, engineers have been instead increasing the number of transistors in a chip.

¶6 · There are more differences between the theoretical and the practical view of Turing completeness. For example, the syntactic requirement for completion —the syntax of a complete language has to be infinite— is theoretical, and then it is independent of the use and the implementation of the complete language. From the practical point of view, we can recommend that the syntax of a complete language should be a recursive set, in the sense of Post (1944). Following this recommendation, syntax parsing is decidable, that is, the function implementing syntax parsing is total, so given any string of symbols there is a program that always determine whether or not the string of symbols belongs to the complete language. This is just a first recommendation, and it still leaves open many possibilities for the syntax of a complete language: should it be context-sensitive, or context-free, or regular, or even something in between? Well, it will depend on the use of the complete language, and also on its implementation, which includes the implementation of its functional semantics. For example, Lisp is a complete language created by McCarthy (1960), and the syntax of Lisp is context-free. While the mathematical requirement is absolute, engineering recommendations are concerned with applicability, so they have to consider its use and implementation.

§2.5 We are Turing complete

¶1 · Our species, *Homo sapiens*, is Turing complete, meaning that each individual member of our species is Turing complete. There are many ways to show that we are Turing complete. The direct way is to show that, given the table of any Turing machine, we can imitate the computations of that Turing machine on whatever data, save for time or tape limitations. And that is what we can do when we understand how Turing machines work, and then understanding Turing (1936) proves that we are Turing complete. In fact, it is enough to understand the sections of the paper where the Turing machine is explained, §2 and §3, or an equivalent text as for example §2.1 and §7.1 here.

¶2 · Understanding any complete language shows that we are Turing complete, too. Take, for example, Lisp. Lisp is a complete language because the table of any Turing machine can be expressed in Lisp, see §7.1, so any Turing machine can be imitated by a computer running a Lisp interpreter, save for time or tape limitations. Now, to understand the Lisp manual by McCarthy et al. (1962) means that we can calculate the result of any Lisp program by ourselves. This, being Lisp a complete language, means that we can calculate any recursive function given to us as data (the Lisp program), save for time or tape limitations. This argument can be applied, *mutatis mutandis*, to any other complete language.

¶3 · The fact that the Lisp manual by McCarthy et al. (1962) is written in English shows that English is a complete language if Lisp is a complete language, which it is, because it means that understanding English is enough to understand Lisp. Again, we can apply the same argument, *mutatis mutandis*, to any pair of complete language and natural language. See that if a natural language is complete, then we can explain any complete language in it, but that if a natural language is not complete, then no complete language can be explained in it. This shows that most, if not all, natural languages are complete. In any case, assuming that any human child will acquire English if raised in an English speaking community is assuming that we are Turing complete. And again, this argument can be applied, *mutatis mutandis*, to any other complete natural language.

¶4 · The fact that we are Turing complete has many consequences, and some of them will be developed below, but by now you should already understand what is Turing completeness, and to look for some of the consequences by yourself is an enlightening exercise that I strongly recommend you. So, please, pause a bit, and explore it by yourself!

§3 Evolution

§3.1 Software is much cheaper than hardware

¶1 · As we have seen in §2.1, computing by Turing (1936) distinguishes software from hardware, where software is what is written on the tape, and hardware is the computing device. Again, we are most interested in real computing, so we need to translate hardware and software from Turing's computing to real computing.

¶2 · Hardware is the computing device, or the matter the computing device is made of, which is just matter. Actual real computing devices can be made of silicon, as for example the Intel 4004 microprocessor, see Betker, Fernando & Whalen (1997). But they can also be made of other materials, as organic matter in the case of the brain, or metal and wood in the case of the mechanical computers of Babbage. We will assume here that, in the general case, to deal with hardware is to deal with matter. Hardware is matter seen from the computational point of view.

¶3 · Software, or data, is what is written on the tape, which is just memory. So we should investigate writing on real tapes and in real memories. And, for example, if the real tape is a binary magnetic tape, then we only need two stable magnetic states, which can be the two ends of the hysteresis loop of the material. In this case, some energy is needed to change the state. There are some other types of real memory that require energy to keep what is written, as for example random-access memory (RAM) used in personal computers built with transistors and capacitors, where the charge in the capacitor stores the information. As no real capacitor is leak-free, some energy is needed to keep its charge. We will assume here that, in the general case, to deal with software is to deal with energy. Software is energy seen from the computational point of view.

¶4 · To transport software, as for example the programs and data that make an application, we can use light carried by an optical fiber cable, where light is emitted and absorbed in photons, which are massless elementary particles. This shows that we do not need to move matter to transport software, and that moving energy is enough. Of course, to transport hardware we have to move matter. Again, from the point of view of computing, matter is hardware and energy is software.

Matter · Hardware
Energy · Software

¶5 · The physical relationship between matter and energy was given by Einstein (1905):

$$\mathrm{E} = mc^2.$$

In this very famous equation, E stands for energy, m for mass, and c denotes the speed of light. As c is big, c^2 is much bigger, meaning that a tiny amount of mass can be transformed into a huge amount of energy, as shown by an atomic bomb exploding, or by a nuclear power station generating electricity. In this case, software is much cheaper than hardware, where 'much' means 'c^2 times'. Then, the power of a Turing complete device transforming hardware computations into software computations is somehow related to the power of an atomic bomb transforming matter into energy. It is perhaps only a coincidence that the first full programmable computers were used to calculate the first atomic bombs.

¶6 · Of course, to build a computing device we do not need to transform pure energy into matter, because just by rearranging some already existing materials we can construct a computing device, but even rearranging matter is more expensive than rearranging energy. Thus, hardware is a matter arrangement, software is an energy arrangement, and Turing completeness is the capacity of some matter arrangements to mimic any matter arrangement by taking an energy arrangement as input.

¶7 · In any case, software has to be executed by hardware, so in the case of a Turing complete device we have the following progression. For one function, for example for addition, an adder machine is cheaper than a full programmable computer with a program for addition. Even for the four arithmetic operations —addition, subtraction, multiplication and division—, a basic calculator is cheaper than a computer with a calculator program. However, beyond some point, if software is much cheaper than hardware, the full programmable computer wins. And the fact that there are now much more personal computers than calculators proves that software is much cheaper than hardware.

¶8 · More formally, we can compare the cost of n computing devices, where each one calculates a function, with the cost of a Turing complete computing device plus the cost of the software needed to calculate those n functions, and, if software is much cheaper than hardware, then increasing n there will be a point beyond which the second is cheaper. If the cost of building a computing device i is H_i, the cost of building a Turing complete device is U, the cost of the program i that imitates the computing device i in the Turing complete device is S_i, and $S_i \ll H_i$, then for some number m, which we will call *the breaking point*:

$$\sum_{i=1}^{m-1} H_i \;\leq\; U + \sum_{i=1}^{m-1} S_i \;,$$
$$\sum_{i=1}^{m} H_i \;>\; U + \sum_{i=1}^{m} S_i \;.$$

And therefore, beyond the breaking point, that is, for any $k > m$:

$$\sum_{i=1}^{k} H_i \;\gg\; U + \sum_{i=1}^{k} S_i \;.$$

¶9 · As soon as it was possible to build full programmable computers for a price that was below a critical point, they won over non-programmable calculators, because software is much cheaper than hardware. Just by using the software to imitate a calculator, a full programmable computer can do whatever the hardware calculator can do, because the full programmable computer is Turing complete. While the calculator can only perform a fixed set of operations, those that are built in its hardware, the same Turing complete hardware, by using some other software, can compute any computable function, including calculations sequencing the basic four operations. And that was just an example, the complete field of application of full programmable computers is too broad to be detailed here, and in fact our modern society depends too much on they. Full programmable computers pervade our modern society.

§3.2 Computing is for problem solving

¶1 · Mathematics in not concerned with the use of its models, and this applies to Turing's computing model, too. But we are interested in the rôle that computing plays in evolution, so we must answer a question: What is the use of computing?

¶2 · Computing was founded by Turing (1936) to serve as a mathematical model of problem solving. Turing defines his machine to prove that the *Entscheidungsproblem*, which is the German word for 'decision problem', is unsolvable. After defining the Turing machine, he shows that there is not any Turing machine that can solve the problem. However, this proof is valid only under the assumption that the set of Turing machines exhausts the ways of solving problems, where each Turing machine is a way of solving, because then that no Turing machine solves a problem implies that there is no way of solving it. This assumption is Church's (1935) thesis reformulated for problem solving by computing.

¶3 · For now, we can forget about Church's thesis, because it is just a assumption that mathematicians need to perform a proof, but we should keep in mind these two key ideas:

- computing is for problem solving, and
- each Turing machine is a way of solving,

where a Turing machine implements a recursive function.

¶4 · That a function is a way of solving makes sense, because the function is the mathematical abstraction for transformations, so to go from a problem to its solutions a function is needed. Therefore, from the problem solving point of view, we will call a recursive function a *resolution*. As any recursive function is a resolution, and a generic solution can be the solution to any problem, then, given a problem, what we need is not any resolution, but a valid resolution for that problem. A *valid resolution for a problem* is a function that returns exactly the set of the solutions to that problem. Usually, a solving resolution is enough, where a *solving resolution for a problem* is a function that returns some of its solutions, only solutions and at least one. Note that, if problem π has at least one solution, then a valid resolution for problem π is a solving resolution for problem π.

¶5 · A Turing machine is then a resolver, because it implements a recursive function, which is a resolution. In other words, if Turing machine $\mathcal{M}$ implements a solving resolution for problem π, then it returns at least one problem π solution, and it does not return any non-solution of the problem π, so we can take anything that Turing machine $\mathcal{M}$ returns to solve problem π. Thus, we say that Turing machine $\mathcal{M}$ solves problem π, and that Turing machine $\mathcal{M}$ is a way of solving problem π.

¶6 · In this context, a Turing complete resolver is a resolver that can be programmed to solve any problem that any resolver can solve. As seen in §3.1, beyond some critical number of problems, if software is much cheaper than hardware, then a Turing complete resolver will be more profitable than the equivalent set of hardware resolvers.

§3.3 Turing completeness is evolutionarily disruptive

¶1 · Seeing brains as computing devices, the advantage of a Turing complete brain against one that is not Turing complete is also huge beyond the breaking point, and because of the very same reason: implementing a function in software is much cheaper than implementing the same function in hardware. However, to see it clearly, we should see it from the point of view of problems.

¶2 · Life can be assimilated to the survival problem. From this point of view, which is the problematic view of life, evolution is a problem resolver. Evolution solved some subproblems of the survival problem by designing systems, as the cardiovascular system, and it solved their sub-subproblems by designing organs, as the heart.

¶3 · But evolution cannot solve the problems faced by individuals in their day-to-day living, because those problems depend on casual circumstances. For solving those problems faced by individuals, evolution designed the nervous system, the brain organ, and even the specialized nervous tissue and neuron cell. Then, broadly speaking, the function of the nervous system is to deal with information, and the function of the brain is to take information from the body, compute what to do in order to resolve according to the circumstances, and send the resulting command information back to the body. In other words, the brain is the resolver of the problems of the individual.

¶4 · There is a delicate interaction between the brain and the rest of the body, which is calibrated by the proper distribution of responsibilities between the two problem resolvers involved, evolution and the brain. For example, heart beat rate can be autonomous, as shown by a separated heart beating, but the brain can command to increase the beat rate when running, for example, to flee from a predator.

¶5 · Not all living individuals are Turing complete, so we might wonder what difference does this make. A resolution is a function that takes a problem and returns solutions, right and wrong solutions. So, being a function, a Turing complete individual can express and calculate, that is, imagine, any resolution in his complete language, while a more limited individual will apply its limited set of resolutions to any problem. The key point is that a single Turing complete individual can imagine any possible way of solving a problem, and then he can execute any of the imagined resolutions that returns right solutions, while an individual that is not Turing complete can only apply those resolutions that are implemented in the hardware of its body, mainly in the hardware of its brain.

¶6 · Species that are not Turing complete need a genetic change to modify their set of resolutions, while Turing complete individuals can apply new resolutions without any hardware change, but just by a software change. So the timing of creativity depends on evolution until Turing completeness is achieved, and it does not depend on evolution after that point for the species that achieve Turing completeness. We will call every point where an evolutionary path achieves Turing completeness an *evolutionary singularity*. In other words, an evolutionary singularity is any evolutionary moment when the brain surpasses evolution in problem solving. You can find another explanation in annex §7.2.

¶7 · In summary, creativity is slow until an evolutionary singularity and creativity explodes after every evolutionary singularity because, after achieving Turing completeness, performing new ways of solving the survival problems becomes cheap and available to single individuals while, before achieving it, any single new way of solving them required genetic changes on species over evolutionary time spams. Thus, Turing completeness is evolutionarily disruptive.

§3.4 We are the only Turing complete species

¶1 · Being disruptive, it should be easy to distinguish a Turing complete species from one that it is not Turing complete. Now, we will list some characteristics that any Turing complete species should show. Our aim here will be to find some Turing complete species other than us.

¶2 · Each ecological niche can be seen as a survival problem, because each niche is defined by the conditions that a species has to satisfy to survive in it. Then a Turing complete species, which can solve problems universally, cheaply and quickly, can occupy nearly any niche. We can survive in nearly any climate because if, for example, it is cold, then we resolve to use cloths. And similarly for nearly any other condition. A Turing complete species can survive in very different niches.

¶3 · A tool is a device that facilitates a transformation. Then, from the problematic point of view, a tool is the physical implementation of a resolution, that is, tools are realizations of resolutions. For example, when a problem solution is a wood object, then the resolution can consist in cutting some wood, which we can only perform with a tool such as a saw. A Turing complete individual can imagine and build tools to solve problems and even to build tools, when a tool is considered the solution of a subproblem. A Turing complete species uses tools extensively.

¶4 · A Turing complete individual, when facing a new niche where it is easier to transform the environment than to transform its body, will resolve to modify the environment. As this will be usually the case, Turing complete species will perform niche construction extensively, creating a variety of artificial ecosystems.

¶5 · And we should not forget the characteristic that defines Turing completeness: A Turing complete individual can be programmed, or instructed, to calculate any recursive function in his complete language. Failing to calculate just one recursive function is disqualifying. For example, if a species cannot learn to count up to any number, then that species is not Turing complete. Note that the converse is not true, so a species that counts is not Turing complete if it fails to calculate any other recursive function. There is a longer list of Turing complete characteristics in §4.

¶6 · Our species, *Homo sapiens*, qualifies easily on all of the characteristics, but it seems that no other species qualifies. No other species has an external complete language for communication, though it could be that a species had an internal complete language for thinking about which we were not aware. However, if that were the case, then that species would show creativity in solving its problems in different ways, in occupying several niches, in using tools extensively, and in creating a variety of artificial ecosystems. Given that no other species shows a level of creativity in the same order of magnitude than us, our conclusion is that we are the only Turing complete species.

¶7 · To my knowledge, no other species is as creative as ours, by far. Even our closest relatives ever, Neanderthals, keep behaving the same way for more than two hundred thousand years, according to Fagan (2010), page 80. If this is true, then our best guess is that *Homo neanderthalensis* was not Turing complete, and that the only Turing complete species ever is our species *Homo sapiens*.

§3.5 Why only us?

¶1 · Turing completeness is evolutionarily disruptive, but it is not evolutionarily easy or otherwise there would be other Turing complete species with us. This raises the question: Why only us?

¶2 · The main reason is that evolution has not foresight. Evolution works as a short-sighted enterprise that asks its executives that every step has to be profitable. Taking always the easiest next step constrains very much the possible paths to anywhere, and also to Turing completeness. Our task here will be to find a point in the evolutionary landscape from where the path of least resistance goes to Turing completeness.

¶3 · Firstly, the point should be in an area of enough complexity. We saw in §3.1 that Turing completeness is advantageous only beyond the breaking point. This means that Turing completeness is not for every species, but only for those that face a number of problems that is greater than the breaking point. We have not determined the breaking point for evolution precisely, but we can assume that insects are below that point, and that apes are above it.

¶4 · Secondly, Turing completeness requires a language, and only cooperative species can benefit from using one, as argued by Tomasello (2008). Cooperation happens whenever a problem is shared by two or more individuals that are resolving it together. In cooperation, communicating true information is useful, while in competition it is not. So the evolutionary path to Turing completeness has to pass through an area where cooperation is required. Cooperation can be a stringent condition because natural selection is basically competitive, as shown by the subtitle of its founding book by Darwin (1859): "On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life". However, although it took some time to explain scientifically the evolution of cooperation, see Axelrod & Hamilton (1981), cooperation is certainly possible, as shown by multicellular organisms, and by honey bees.

¶5 · Taking together both conditions, cooperation and complexity, they seem to filter out all species but one, ours. A cell is too simple to be Turing complete, so cell cooperation in multicellular organisms is not enough. The same is true for honey bees; insects are too simple to take advantage of Turing completeness, so the cooperative honey bees never evolved a complete language. On the other hand, chimpanzees are complex enough, but they are not cooperative enough, see Tomasello (2008) §5.1.1, so they did not evolve Turing completeness, either.

¶6 · But, for some unknown reason, some of our ancestors were in a situation that fostered cooperation. As chimpanzees do not cooperate, this happened after the chimpanzee-human split, about six million years ago according to Patterson et al. (2006), or it caused the split. Then, a new species started to evolve cooperative traits, including cooperative communication. From that point, the evolutionary path went down enhancing cooperative communication until a protolanguage first, and then a complete language was achieved. If even Neanderthals were not Turing complete, my guess is that Turing completeness was achieved in the time frame of the anatomically modern *Homo sapiens*, but before the African exodus, so from 60k to 200k years ago.

¶7 · New behaviors related to the solution of problems, such as feeding, mating, and surviving, should proliferate after achieving Turing completeness. And being a tool the physical realization of a resolution, then an explosion of new tools should start whenever an evolutionary singularity happens. So the archaeological record of human tools should

point to our evolutionary singularity, that is, to the moment when we achieved Turing completeness.

§4 Humanity

§4.1 Uniqueness

¶1 · Every species is unique. Nevertheless, if Turing completeness is evolutionarily disruptive and we are the only Turing complete species, as I am arguing, then we are "more unique" than other species. At the very end of §2, I encouraged you to investigate the consequences of being Turing complete, and now I will do it myself: in this section §4, I will explore our human uniqueness, as it results from being Turing complete. However, before going to the details, two warning notes are in order.

¶2 · This section §4 is more tentative than the others, and then it should be read in a different mode. My main intention here is to show that many concepts that are completely alien to science from other points of view are at least addressable from the problem solving point of view of both evolution and computing presented in this paper.

¶3 · The politically correct position is to say that our species is not fundamentally different from other species, and here I am arguing otherwise. But I honestly believe that we are the only Turing complete species and that this is the cause of our too much dominant position in nearly every ecological niche on Earth. Seeing it objectively, we are a pest for nearly every other species, causing the extinction of many of them. In this case, political correction defending that we are just a species like any other can be used to ignore our responsibilities, while understanding our unique position among all species can contribute to resolve the global problems that we are causing. Amen.

§4.2 Creativity

¶1 · Turing completeness is defined by a single condition: pan-computability. A universal Turing machine, which is the prototype of Turing completeness, is every Turing machine that can compute whatever any Turing machine can compute, but no more. This means that to perform any specific computation you can, either build the Turing machine specific for that computation, or write the program for that specific computation on a universal Turing machine. Either way the outcome of the computation will be the same, and the only differences would be that the first can run faster, and that the second can be implemented faster, once you have a universal computer. The second is better for modeling, because writing software models is much faster and much cheaper than building hardware models, but, again, the results of the computations are the same.

¶2 · In other words, creativity is the only exclusive feature of Turing complete problem resolvers. And this explains an elusive fact: every time a specific behavior is presented as uniquely human, it is later rejected when it is found in another species. The point is not that we behave in some specific way to solve a problem, but that we are free to imagine any way to solve our problems. Creativity is the mark of Turing complete resolvers.

§4.3 Freedom

¶1 · Freedom is the source of creativity, because without freedom one cannot be creative.
¶2 · When facing a problem, being Turing complete makes a whole difference. It means that one is free to imagine and to perform any resolution. That is, anything that can be expressed in the complete language of the Turing complete individual can be examined, evaluated, and executed. Well, some resolutions cannot be executed but, if something impossible is tried, then something was wrongly evaluated.
¶3 · This freedom contrasts with the fixed ways of solving their problems of non-Turing-complete resolvers. As in the case of a basic calculator, where you can only select one out of a fixed set of operations, their ways of solving are fixed. As long as those fixed ways solve all the problems they face, they will survive, and all surviving species but one are in this situation. However, though some of them show a variety of behaviors, they lack the freedom we have.
¶4 · This is a good moment to note that Turing complete problem solving is more open and then more difficult than more restricted forms of problem solving, because Turing complete problem resolvers have much more chances to be wrong. We can also say that Turing complete problem resolvers are free to be wrong. Therefore, as only we are free to be wrong, or right, only we are responsible. We cannot put the blame on others.

§4.4 Language

¶1 · To have a complete language and to be Turing complete is the same thing.
¶2 · An important mathematical property of complete languages is full reference, see §2.3, which includes self reference: any of the complete language objects can be referred from within the complete language. This was firstly shown by Gödel (1930) when he was able to assign a different natural number, now called the Gödel number, to any arithmetic expression, or sequence of expressions, including the natural numbers themselves! In the case of natural language, the word 'Bible', for example, refers to a whole text, and the sentence 'this sentence is false' is a statement about itself. By the way, that sentence shows the funny things that self reference can express. But what is important here is that we can refer to anything that can be said the same way that we refer to any real object. Using arbitrary sounds to refer to real objects is something that dogs can already do, as shown by Pavlov (1927), so our ancestors surely had this capacity before reaching Turing completeness. And now, in our complete language, we can think of imagined concepts the same way we think of real objects. Language exceeds reality.
¶3 · That our language is complete means that we can express and calculate any recursive function in it, that is, we can imagine any algorithmic rule for change in our complete language. So in our complete language we can imagine any possible change whatever, whether it is happening or not. A complete language can then describe what is actually happening, what could possibly would have been happening, and even the impossible happening. Changes are objects of the complete language. Language exceeds actuality.
¶4 · Taking both properties together, full reference and change representation, they extend our world enormously. By the first, we can think about any complete language object, and by the second, changes are complete language objects. Therefore, as language objects can be real or imagined, we can refer to any object, real or imagined, and to any change, real or imagined, in our complete language. If actual reality has three spatial dimensions, then our world as extended by our complete language has two additional dimensions: possibility or freedom, and change or time.

§4.5 Will

¶1 · If you were an engineer who had to add Turing completeness to an already working species, where would you add it? Turing completeness is so powerful that you can entertain the idea that it can be added anywhere, but a little thinking can persuade you that it would be more profitable to use it where it is more needed, that is, Turing completeness should help to resolve the more important problems for the individual. Fortunately, every complex species needs, and has, a system for prioritizing problems. So, on top of perception, a complex species needs attention, to be aware of urgent threats, and it needs intention, to decide how to face the most urgent threat. From this cursory analysis, it seems that the best way to proceed would be to implement a Turing complete intention.
¶2 · In the case of a non-Turing-complete species, intention is reduced to choose one resolution (one button) out of a finite set; for example, flee or fight when a predator is coming. That decision can be made using the weighted sum model, for which neural networks are specially suited. A weighted sum implementation of intention is efficient and it gives its results very quickly.
¶3 · The process is different when we implement a Turing complete intention. The problems that attention prioritize are then translated to the complete language where possible resolutions are generated and compared until one that is considered the best is executed. Translation: The problems that need attention are verbalized and possible plans are devised, examined and evaluated until one is judged fit for execution. A verbalized implementation of intention is flexible, but it does not gives its results quickly.
¶4 · We can get the best of both by keeping the efficient weighted sum intention and adding the verbalized intention to it, so when a decision is needed quickly, weighted sum intention is used, and when time is not pressing, verbalized intention is chosen. This can be implemented in parallel, working both simultaneously, and when a decision is needed the verbalized one is preferred, if available, but if the verbalized decision is not yet available, then the weighted sum one is used. In other words, at first the weighted sum decision is adopted, but if a verbalized decision is available later, then the verbalized decision vetoes the previous one.
¶5 · Intention is not free. Intention is related to problem resolution, which all complex enough species have to implement in order to survive. The result of adding verbalization in the complete language to intention is will. Will is free.

§4.6 Explaining

¶1 · To explain why a person did something, we have to show in detail the intentional process that resulted in his behavior; in other words, we have to uncover his will. So, firstly we should describe which problem he was facing, that is, to which problem he was attending. Sometimes this can be assumed, but it would be impossible to understand what he did if we do not know what problem he was trying to solve by doing what he did. And sometimes the motive is the only important point, but more frequently the way is also important, and then we should list the possible plans he could had followed, and we should show that the plan he actually accomplished was the best for him to undertake given his circumstances, from his point of view.
¶2 · We can use the same schema to explain other complex living beings behavior, because they have attention and intention, and other simpler living beings behavior, because they face problems that they solve using resolutions designed by evolution. In the case of

evolution the problem is the survival problem, always, so it can be omitted, and the evolutionary ways of solving are those described by Darwin and other biologists.

¶3 · Currently, explaining non-living events is easier, but less convincing. A scientific explanation is indeed a impoverished explanation, because we assume that non-living entities do not face problems. From this assumption, it follows that for non-living entities neither problems nor solutions have any meaning. What about resolutions? Well, a resolution is a program, an algorithmic rule, so in this case the resolution is the law of nature that rules what happens. Then, the scientific explanation just says that, given the law of nature and the state of things, the only possible outcome was what it happened. For science, nature is the simplest resolver, that is, nature is a mechanism that behaves always and unconditionally the same way. Although they were surely less useful, the explanations of natural events were more convincing when gods governed nature!

¶4 · We see other living beings as resolvers, that is, as pursuing their goals, and ultimately fighting for their survival. We also see evolution as pursuing a goal, producing surviving beings, even though we know that evolution is blind so it only follows the path of least resistance, as a projectile following its ballistic trajectory. And when we ask 'what is the universe for?', we are assuming that the universe is also pursuing its goal, a mysterious goal. To me, this shows that we can only explain what happens from the problem solving point of view by mirroring the way we solve our problems.

§4.7 Understanding

¶1 · From the problem solving point of view, a question is the expression of a problem, and its answer is the expression of its solution. In some languages, the type of question restricts the answers. For example, in English a who-question asks for a person. Now, it is interesting to see that how-questions ask for resolutions, and then a resolution is a solution, and specially that why-questions ask for explanations.

¶2 · As seen in §4.6, an explanation is a discourse, or text, in a complete language that, given a solution, describes and justifies both the problem that the solution solves and the resolution used by the resolver to get the solution. An explanation can be unconvincing if the justifications are unconvincing: why was that problem and no other problem attended, why was not other resolution devised, why was this resolution judged better than that other, and more. Each of those why-questions would ask for its corresponding explanation, resulting in a potentially unending tree of explanations, and of why-questions.

¶3 · The tree shape is typical of problem solving. Usually, a complex problem can be decomposed in several, hopefully simpler, subproblems and some of these can be further decomposed, producing a tree of related subproblems. Here, we will not follow more deeply the convoluted shape of problem solving, see the details in Casares (P), but the conclusion is that in order to express problem solving fully, with all of its questions, answers, and explanations, a complete language is required. More precisely, full problem solving requires a functional semantics on a tree-structured syntax, see Casares (S). Note that simpler ways of problem solving, as for example trial and error problem solving, do not need any language at all.

¶4 · In between there could be semi-complete languages in which semi-explanations are possible, but here I will assume that evolutionarily their locations are near and uphill Turing completeness, so they are always in the path to a complete language. To show that this assumption is sensible, we should compare our natural language with other animal communication systems. All other animal communication systems are closer to the finite

set of buttons of a basic calculator that are needed to choose the operation to apply than to the textual programs that command the operation of a full programmable computer, which closely resemble our natural language. That is, all other animal communication systems use expressions that are fixed in meaning and that cannot be combined to produce more complex expressions. My conclusion is that there is not any semi-complete language in nature. Of course, in our own evolutionary path, it is very possible that some of our ancestors spoke a semi-complete language.

¶5 · The conclusion is then that explaining requires a complete language. Semi-explaining requires a semi-complete language, but there are not semi-complete languages in nature, except temporarily, and there is none currently, so we can forget about semi-complete languages here. Therefore, only Turing complete resolvers can explain what happens, and understanding requires a complete language.

§4.8 Consciousness

¶1 · Explaining and understanding are the sending and the receiving end, respectively, of the semantic communication channel. If I am right, then both ends are implemented in human intention, or will, where the semantic communication is done in a complete language. Then, will is a verbalized intention that we experience as an inner voice, or an internal dialog, called the stream of consciousness. Complete language properties, mainly full reference and change representation, determine the properties of conscious thinking.

¶2 · A resolver is an agent of change. Therefore, a language that can refer to change is needed to explain a resolver. As representing change seems to be an innovation of complete languages, only Turing complete resolvers can explain themselves to themselves. This conscious thinking about one-self is called self-consciousness.

¶3 · Turing complete resolvers can explain any resolver whatever, including themselves and other Turing complete resolvers, in their complete language. Therefore, a Turing complete resolver can elaborate theories of mind, that is, models of wills, of his own will and of others wills.

¶4 · In fact, because of full reference, Turing complete resolvers can explain any agent whatever in their complete language. Then, Turing complete resolvers can elaborate theories of anything because, at last, any computing device can be modeled exactly by a Turing complete computing device. The possibility of mirroring in our complete language anything that happens, including our own thinking, evokes easily the concept of consciousness as reflection.

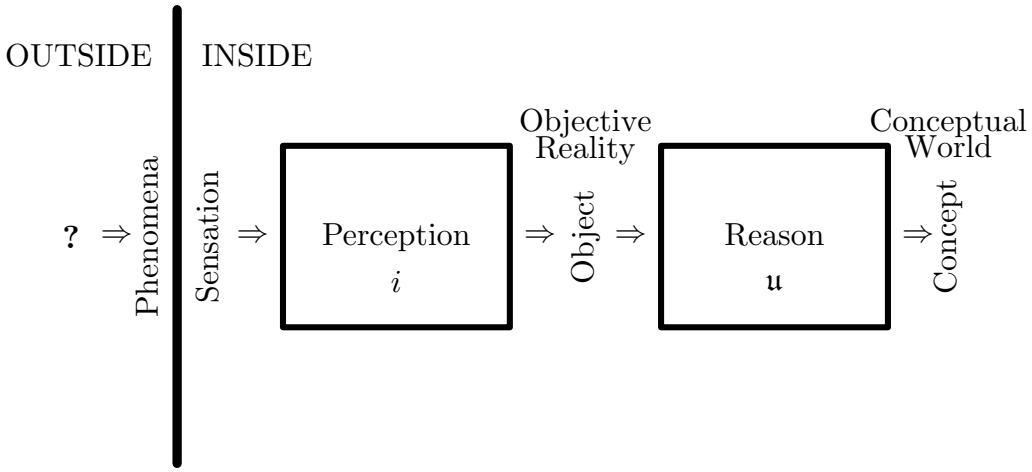
¶5 · By the way, the semantic communication channel was excluded by Shannon (1948) on purpose from his mathematical theory of communication, as he explains in the second paragraph of the introduction, page 379. This was possibly necessary to state a mathematical theory, but, to me, it is not entirely true that “These semantic aspects of communication are irrelevant to the engineering problem.”

§4.9 Subjectivism

¶1 · The theory proposed here is founded on the distinction between software and hardware, which is the computational version of the physical one between energy and matter, as we saw in §3.1. And both are closely related to Descartes' dualism based on the fundamental distinction between mind and body. Body, matter and hardware are perceivable and permanent objects, while mind, energy and software are the agents that move, transform and drive the objects.

¶2 · These are the kind of explanations that would result should our cognition were basically a Turing complete intentional layer built on top of perception, as I am arguing here. Then, perception would present the objects composing the actual reality, and the intentional Turing complete layer, being able to represent possibility and change, would reason about possible movements, transformations and modifications of the perceived objects. To me, the dualism of Descartes, or the disagreement between Parmenides and Heraclitus, shows the workings of our cognition, and in particular that our cognition was developed by evolution in two main stages.

¶3 · If I am right in this, then the equivalence of energy and mass of relativity and the wave-particle duality of quantum mechanics are two signs showing that our cognition is limited and distorting. Nevertheless, the noumenal warning by Kant should be attended: what is out there is out of our reach. And this, together with the truism that our cognitive machinery is as it is, implies that all of our knowledge is subjective. This just means that all we know is as it is because our cognition is as it is. A triviality, isn't it?



§4.10 Philosophy

¶1 · As our complete language is external, we can use it to communicate with other Turing complete resolvers. For example, a cooking recipe is just the list of rules that should be followed to transform some raw ingredients into an edible dish, and a cooking book is a list of cooking recipes that anyone who understands the complete language in which it is written can follow. The point is that, being Turing complete, we can be instructed to follow whatever rules.

¶2 · The core of being Turing complete is being able to be instructed, or programmed, to follow whatever rules. This seems nice, but it can also be ugly, because it means that we can be programmed to do anything, whether it is good or bad. To decide which rules are good and which are bad is the realm of ethics and morality.

¶3 · The legal system of a state, which is the list of rules that anyone living in the state has to obey, is not possible without Turing complete citizens speaking a complete language. Note also that the state is not a real object, but an imagined concept. To decide which rules are good and which are bad for the state is the realm of law and politics.
¶4 · Because we are Turing complete, we assume that reality is governed by algorithmic rules. Then, the task of physics, and of science generally, is looking for the laws that rule reality, or, in other words, is translating reality to language. To determine which rules govern reality is the realm of physics and science.
¶5 · Engineering and medicine try to take advantage of physics and science to resolve practical problems. To determine which rules resolve our practical problems is the realm of engineering and medicine.
¶6 · If we are interested in the consequences of applying any rule system whatever, without any constriction, we should study mathematics. To determine what any rule system entails is the realm of mathematics.
¶7 · Mathematics, engineering, medicine, physics, science, law, politics, ethics, and morality, that is, everything that was called philosophy in antiquity, is characteristic of Turing complete individuals.

§4.11 Culture

¶1 · Our ability to deal with rules extends beyond philosophy. For example, we can imagine entire worlds to whatever level of detail in our complete language and, when the imagined world is fictitious, the result is literature. If the set of rules is not imagined, but designed to be fun, or just competitive, then the result is a game or a sport. The *Homo ludens* concept by Huizinga (1938) comes from here, because only a Turing complete species can enact and follow arbitrary systems of rules.
¶2 · This leads us to culture, which is very close to play, or as Huizinga (1938) has put it, page 46: "In the twin union of play and culture, play is primary." Nevertheless, young children and other species play is imitative, and imitation is not creative. That is, primary play does not follow arbitrary rules.
¶3 · A specific culture is the system of norms that prevail in a specific society. When it is used in general, culture is the set of norms found in societies. Then, for example, Shinto is part of the culture of Japan, and philosophy is part of culture. The characteristic of a cultural norm is that it is learned, and then every cultural norm is a rule, but not every rule is a norm. For example, a law of nature, as gravitation, is a rule that every society follows, but that it is not part of culture because it is not learned. The same happens to the rules that govern bees societies because, being genetically coded, they are not learned.
¶4 · Culture is not directly related to socialization, but to learning.

- Bees live in very cohesive societies with individual specialization but they have no culture because they cannot learn.
- Chimpanzees live in sparse societies without individual specialization and they have a primary culture because they are good imitators.
- We modern humans live in very cohesive societies with individual specialization and we have full culture because we are Turing complete.

¶5 · However, individual learning is not enough to have a culture. If the rules learned by an individual are only for himself, then his society cannot incorporate those learned rules into its culture. To develop a culture, the learners in the society have to include

imitation in their set of resolutions. So, the minimum for a culture seems to be a society of imitative learners.

§4.12 Artificial worlds

¶1 · Culture creates an artificial normative world on top of both genetic codes and the laws of nature. And this new level of rules on the top has consequences that fall down to the other levels: an increasingly cultured Earth is causing the extinction of many species when they are deprived of suitable ecosystems, and we are starting to hack the genetic codes with consequences that are difficult to foresee. We should be optimistic, but only because the pessimistic view is too bad to hold sanely.

¶2 · As it is usually easier to modify the environment than to modify our body, we usually prefer to change the environment to fit our bodies, rather than doing the converse, see §3.4. So we fit the environment to our bodies, rather than fitting our bodies to the environment by restricting ourselves to those environments to where our body is fitted. This results in an intensive use of tools, such as fire and clothes, for example. Domestication of plants and animals should be understood in the same way: domestic plants and animals are used as tools. And it results also in the modification of lands and seas to fit our necessities in always increasing areas, occupying and modifying nearly every ecosystem on Earth. For a Turing complete resolver, everything is a tool.

¶3 · We saw in §3.3 that Turing completeness is evolutionarily disruptive because the rate of change of evolution is very slow compared to that of Turing complete individuals. In other words, evolution cannot follow our quick ways of doing. And then, when we modify, sometimes to null, nearly every ecosystem for whatever reason we think we need, many of those other species that were surviving on those ecosystems get extincted. We are causing a massive extinction. But our own survival depends on the survival of life, so we cannot keep going as we are currently going. We are responsible for the whole life on Earth.

¶4 · In summary, we are the dominant species on Earth because Turing completeness provide us with a great power, and with great power comes great responsibility.

§5 Discussion

§5.1 Realization

¶1 · Before discussing, we should discuss what to discuss about. And I will not discuss about §1, because it is just the introduction, that is, some text paving the way for what follows. Regarding §2, most of it is mathematical or just factual, and then not amenable to discussion. Perhaps the only question open to discussion in §2 is why I have chosen the computing theory by Turing instead of any other.

¶2 · The reason could be that it was Turing (1936) who introduced the key concept of this paper, which is universal computing, also known as Turing completeness, but it is not. I have chosen Turing's computing because it is the most realizable version of computing. As we saw in §2.2, computing by Turing separates cleanly the non-implementable infinities from the realizable computing power, so Turing's computing is real computing, save for time or tape limitations, and these limitations do not refer to computing power. Then, as we saw in §2.4, even Turing completeness is realizable, and in fact there are real Turing complete computers, which can be programmed to calculate any recursive function, save for time or tape limitations.

¶3 · The factual existence of Turing complete computers should prevent any realization problem. And this is mostly the case, except for some philosophers who argue that just running a computer program cannot realize a mind. That deserves some discussion, of course, but I will address it in the next subsection.

§5.2 Meaning

¶1 · Take for instance Searle (1980), page 422, who asks:

> *But could something think, understand, and so on solely in virtue of being a computer with the right sort of program? Could instantiating a program, the right program of course, by itself be a sufficient condition of understanding?*

He answers both with a resounding 'no', and then he gives us the reason.

> *Because the formal symbol manipulations by themselves don't have any intentionality; they are quite meaningless; they aren't even symbol manipulations, since the symbols don't symbolize anything. In the linguistic jargon, they have only a syntax but no semantics. Such intentionality as computers appear to have is solely in the minds of those who program them and those who use them, those who send in the input and those who interpret the output.*

¶2 · I think he is right, completely. A computation by itself has not any meaning. However, if computing is for problem solving, as we argue in §3.2, then the computation has the intention of solving the problem of its resolver. Computers are tools that help us in solving our problems, so in this case *we* are those resolvers who are giving meanings to the computations, as Searle affirms.

¶3 · In the case of living beings, whether they are Turing complete or not, the meanings of their computations can be ultimately traced back to the survival problem, which is the mother of all problems. Of course, Turing complete resolvers can explain and understand what happens in their complete language, while those that are not Turing complete cannot. Nevertheless, if the theory in this paper is right, then intention does not depend on being mindful or mindless, but on being a resolver or not.

¶4 · In summary, computing by itself is meaningless, but computing is for problem solving, and then computing is a means to an end. And this should dispel any doubts on

computationalism, or at least those introduced by Searle (1980 and 1992) and Putnam (1988) that are discussed in Casares (I). Therefore, we fully agree with computationalism: the brain is a computing device.

¶5 · Anyway, the simple picture of a syntax completely disengaged from semantics, as presented by Searle (1980) above, needs a correction for Turing complete computing, because Turing completeness requires a functional semantics, which is a semantics of syntax. Functional semantics is part of syntax because it does not need a resolver to compute it, and it is part of semantics because it gives a precise meaning to some syntactic objects; you can find more details in Casares (S).

§5.3 Problem solving

¶1 · Problem solving is the core of the theory presented in this paper. Problem solving is the nexus that bonds together the mathematics of computing with the biology of evolution and the physics of brains. We have already discussed two links: brains are computing devices, and computing is for problem solving. Taking both together, it follows that the brain is the resolver of the problems of the individual, as seen in §3.3. As brains are products of evolution, we need a link between evolution and problem solving.

¶2 · The link between problem solving and evolution is the evolutionary assumption that solving more problems is sometimes evolutionarily advantageous. Thus, at least on some evolutionary threads, those computing brains that are able to solve more problems have more evolutionary chances. Another general assumption is that, other things equal, solving problems cheaply is better than solving them expensively, so solving problems in software is better that solving them in hardware. The mathematical implications of these assumptions are examined in the problem theory presented in Casares (P), and the evolutionary assumption is confirmed in domestication, as we will see in the next subsection, §5.4.

¶3 · The problem theory in Casares (P) defines a series of five resolvers of increasing computing capacity that exhibits the following property: all problems solved by a resolver are also solved by the next resolver in the series if certain condition is satisfied.

- A mechanism implements a solution; a behavior, for example.
- An adapter implements a set of solutions.
  The adapter condition is that its set includes the mechanism solution.
- A perceiver implements a function from solutions to solutions.
  The perceiver condition is that it implements the identity function i.
- A learner implements a set of functions from solutions to solutions.
  The learner condition is that its set includes the perceiver function.
- A subject implements a function from functions to functions.
  The subject condition is that it implements the universal function $\mathfrak{u}$.

In other words, the subject condition is to be Turing complete.

¶4 · The names of the resolvers in the series try to describe their main properties.

- A mechanism has a body implementing a fixed behavior, so it can only survive in a specific, benign, and very stable environment, as it is the case of some extremophile archaea that act mechanically.
- An adapter has a body that can execute a set of behaviors, so that, in order to adapt itself to the current situation, it can choose one of them by using a trial and error procedure or a trigger condition, as a deciduous tree does.

- A perceiver has a brain implementing a fixed function that translates sensations into perceptions, so it chooses its body behavior on its perceived reality, which is a model of the exterior. Perception translates adaptation to software. A frog is a perceiver.
- A learner has a brain that implements a set of functions, so it can modify its perceived reality in fixed ways. When its current model fails, it selects a new one, thus learning something new about its external environment. A dog is a learner.
- A subject learns in software. A Turing complete subject can reason about any model and about any learning strategy, effectively translating to software the whole problem solving process, because it can compute in software whatever hardware can compute.

A Turing complete subject solves more problems in cheap software than a learner in expensive hardware.

¶5 · This mathematical series defines a resolvers hierarchy that could be seen as a framework for the evolution of cognition. And in annex §7.2, you can find a problem solving comparison between evolution and Turing completeness.

§5.4 Domestication

¶1 · Looking for some confirmation of the relation between brain evolution and problem solving I was lead to domestication, which is the most significant experiment on evolution, particularly on the effects of selection. And one of the effects of mammals domestication is a smaller brain size compared to their counterparts in the wild, see Kruska (1988).

¶2 · As Kruska (2005) has put it, in page 93: "No brain size increase ever occurred due to domestication." His conclusion on domestication, page 100, is:

> *Thus, brain size changes due to domestication must zoologically be evaluated as special adaptations occurring on a species level and being directed to the special 'ecological niche' of domestication even though this description might be considered very broad.*

The question is then: Why does the 'ecological niche' of domestication require less brain? It seems clear to me that the 'ecological niche' of domestication is one in which some very important survival subproblems of the domesticated subspecies are solved for them by us: domesticated animals do not need to look for food because we do it for them, we also protect them from their predators, and we even take care of their reproduction.

¶3 · Therefore, these three assumptions are enough to explain the conclusions reached by Kruska (2005) on mammals domestication and feralization.

- The brain is a problem resolver.
- Brain tissue is energetically expensive; see for example Aiello & Wheeler (1995).
- Constructing is more expensive than destroying, as formulated by the second law of thermodynamics and translated to evolution as the rule of Dollo (1893).

¶4 · Then, when a species is domesticated, the brain tissue that was resolving those problems that were dispensed by us become useless, so animals that have these brain areas less developed will be more energetically efficient, and then they will have more chances of being artificially selected. This means that the energy budgets that are naturally selected in the wild are different that those that are artificially selected under domestication, and that domestication, being an easier niche, favors littler brains because they waste less energy. This, together with the cheapness of destruction, explains the evolutionary quick decrease in brain size of domesticated subspecies.

¶5 · On the other hand, the irreversibility of evolution formulated by the rule of Dollo prevents that feral animals, which are domestic animals that go back to the wild, will

increase quickly their brains. In other words, brain construction should keep the usual evolutionary pace.

¶6 · It is left as an exercise to the reader to explain why the argument for decreasing brains under domestication would not work as well for Turing complete species as it works for learners.

§5.5 Brain size

¶1 · We saw in §3.1 that Turing completeness is advantageous only beyond the breaking point, meaning that Turing completeness requires some complexity. But we also saw in §2.4 that real Turing completeness can be achieved using 2 300 transistors, meaning that Turing completeness does not require too much complexity. Comparing a transistor with a neuron, though neuron connectivity is much greater than transistor connectivity, and given that there are about thirty billion (30 000 000 000 = 30 G) neurons in the brain of a chimpanzee, see Herculano-Houzel & Kaas (2011), it seems that with less than one tenth of millionth of a chimp brain we can already implement Turing completeness. Having such a big margin of error, and though the estimate is very rough indeed, we have to conclude that Turing completeness is not concerned with brain size.

¶2 · Turing completeness can even save some neurons. In a Turing complete brain, non-critical and less frequently used functions do not have to be implemented in hardware, because they could be coded and executed in software, should they were eventually needed. So evolution can abandon the neural circuits that implement those functions, or rather repurpose them, explaining, perhaps, why our brains are equal to or even a bit smaller than Neanderthals brains, see Holloway (1985). In any case, as Turing completeness requires thousands rather than billions of neurons, it is not behind the jump from the 30 G neurons of great apes to the 85 G neurons of humans, Neanderthals or us.

§5.6 Cheapness

¶1 · While §2 is mathematical and §4 is philosophical, §3 is scientific, or empirical. This explains why our discussion here in §5 is mainly about §3.

¶2 · The statement that software is much cheaper than hardware, see §3.1, which is based in the parallel relation between energy and matter, is also empirical. See that, in the $E = mc^2$ equation, the c^2 is big only because of the units of measurement we use, and those units are what they are because we are as we are, as Protagoras would say. In the speed of light units, $c = 1$ and then $E = m$, but life is very slow compared to light. That is, for us one gram of mass is so tiny that we neglect it when we weigh our bodies, but one gram of energy is what exploded in Hiroshima in 1945 killing a hundred thousand people. Nevertheless, in other corners of the universe it could be that 'software is c^2 *times* cheaper than hardware' does not mean that 'software is *much* cheaper than hardware'.

¶3 · Anyway, what is important to us here is that Turing completeness is evolutionarily disruptive because, for evolution on Earth, software is much cheaper than hardware, as we affirm in §3.3.

¶4 · So far we have discussed four of the main thesis of this paper: 'software is much cheaper than hardware' (§3.1), 'computing is for problem solving' (§3.2), 'solving more problems is evolutionarily better', and their consequence: 'Turing completeness is evolutionarily disruptive' (§3.3). The remaining ones, 'we are just Turing complete' and 'we are the only Turing complete species' (§3.4 and §3.5), will be discussed next.

§5.7 Post law

¶1 · The statement 'computing is for problem solving' of §3.2 is mathematical in origin, as it derives from Turing's (1936) intention to prove mathematically the unsolvability of some mathematical problems. However, as we already noted in §3.2, that mathematical proof needs an extra-mathematical assumption, Church's (1935) thesis, which affirms that the set of Turing machines exhausts the ways of solving problems, where each Turing machine is a way of solving, that is, where each Turing machine is a resolver.

¶2 · Note that, save for time or tape limitations, if we are Turing complete, that is, if each of us is Turing complete, then each of us can simulate any Turing machine, and that if each of us is just Turing complete, then the set of problems that each of us can solve individually is precisely equal to the set of problems that Turing machines can solve globally. Then, for Post (1936), and I agree completely with him, Church's thesis is a law of nature stating the limitations of the computing power of our species *Homo sapiens*. Or rather, see Casares (C), Church's thesis is a consequence of Post law: *in computing power, we are just Turing complete*.

¶3 · We should be aware of the difference between the 'we are Turing complete' of §2.5 and Post law: the former says that 'we are *at least* Turing complete', and Post law states that 'we are *just* Turing complete'. So 'we are Turing complete' is an expression of the fact that each of us satisfies the definition of a Turing complete computing device, while Post law goes further and intends to characterize precisely our computing capacity as individuals of the species *Homo sapiens*.

§5.8 Disruption

¶1 · In subsection §3.5, we explain why *only* us are Turing complete: because evolution has not foresight, because Turing completeness is advantageous only beyond the breaking point of complexity, and because Turing completeness requires a complete language, and a language is useful only under cooperation.

¶2 · Both Bingham (1999) and Tomasello (2008) argue that cooperation was the key that started the evolutionary path to our species, and I agree, although cooperation is not enough as shown by other cooperative species that are not Turing complete, see §3.5. As chimpanzees and the other great apes do not cooperate, this happened after the chimpanzee-human split 6 M years ago, or it caused the split. So for some million years there were some species of cooperative hominins, but nothing disrupting happened; in fact all of those species are now extinct except one, ours. We can assume that the number of hominins was between 10 000 and 100 000 most of the time, see for example Hawks et al. (2000), but then something really disrupting happened and thereafter our population has increased dramatically, so it is 7 500 000 000 now (2017) and increasing. A population of 7.5 G is unexpected for a vertebrate, and even more for one as big as us.

¶3 · Bingham's (1999) explanation for the disruption is that, on top of the genetic evolution, our species was involved in an extra-genetic evolution that took place because the brain is a Darwinian processor. This, he says, results in a cognitive explosion, but I dissent: evolution and equivalent Darwinian processors are learners, and a cognitive explosion needs a Turing complete subject, as argued in §3.3 and §7.2.

¶4 · Tomasello (2008) focuses on the differences between us and current great apes, and therefore his explanation is better for times around the chimpanzee-human split than for the cognitive disruption that happened much later. Then, his analysis of cooperation is detailed and convincing, while his explanation of the cognitive disruption is vague.

He refers to recursion as the key, but he believes that no particular hardware devices are needed for grammar, see for example page 281. In this point he is in complete disagreement with Chomsky, see Tomasello (2009), and with me. My point, applicable also to Bingham's theory, is that Turing completeness is a property of some hardware, and therefore some specific brain devices are required to implement a complete language in which full recursion can be expressed and calculated.

§5.9 Recursion

¶1 · Chomsky defends the Universal Grammar theory. The main point of this theory is that language has a genetic component, meaning that language needs some specific hardware devices. This is in agreement with the theory presented in this paper. To me, the best argument in favor of Universal Grammar is that, after putting a chimpanzee baby in the same environment than a human child, the human will acquire language and the chimp will not. But Chomsky seems to prefer the poverty of the stimulus argument, which is based on the observation that children acquire the correct grammars for their natural languages in spite of only hearing a relatively short and surely incomplete sampling of sentences. For Chomsky, Universal Grammar restricts the natural languages that can be acquired. And this does not agree with our theory here, which predicts that any complete language can be learned, but it can be explained by distinguishing learning a language from acquiring a language, as done in Casares (U).

¶2 · Chomsky's belief that language evolved for thinking, rather than for communication, is mainly supported by our theory. Ultimately, a complete language is needed for full problem solving in cheap software, and this agrees with Chomsky's idea. Nevertheless, if cooperation was the catalyst that enabled our evolution to Turing completeness because only cooperative problem solving benefits from communication, then communication was a factor in the evolution of language.

¶3 · Another point where Chomsky's theories agree with the theory presented here is in postulating that implementing recursion was the main innovation of language. Again, the details spoil the agreement. To me, in general, the understanding of recursion in linguistics is biased, see for example Watumull et al. (2014) and Casares (R). And in particular, for Chomsky, recursion is the capability of building a discrete infinity of hierarchical structures, see Berwick & Chomsky (2016) and Casares (M). Meanwhile, for me, the main point of implementing full recursion, which is achieved only by building a Turing complete device, is the possibility of calculating by cheap software whatever more expensive hardware can calculate. Well, I find myself repeating here arguments already made elsewhere in this very same paper, so this seems a good moment to conclude.

§6 Conclusion

¶1 · Why are we so many? Because only we are Turing complete. Turing completeness, which is the capacity of some hardware to compute by software whatever hardware can compute, is the ultimate cause of our success as species, and the argument that supports this answer is based on an empirical fact around Post law and on a consequence that derives from three other laws (or theses).

- Question: Why are we so many?
- Turing: Computing is for solving problems.
- Einstein: Software is much cheaper than hardware.
- Evolution: Solving more problems is evolutionarily better.
- Consequence: Turing completeness is evolutionarily disruptive.
- Post: In computing capacity, we are just Turing complete.
- Fact: We are the only Turing complete species.
- Answer: Because only we are Turing complete.

§7 Annexes

§7.1 A formal Turing machine

¶1 · A Turing machine has four main components, in addition to the tape: the states Σ, the symbols Γ^+, the movements Π, and the table τ.

- The set of states Σ has to be finite, and the initial state `S` has to be in it, $\mathtt{S} \in \Sigma$.
- The set of symbols Γ^+ has to be finite, and the *blank* symbol, written as a dot · so we can see it, has to be in it, $\cdot \in \Gamma^+$. The *blank* symbol represents the null symbol, so it builds empty strings. We will call Γ the set of non-*blank* symbols, so $\cdot \notin \Gamma$, and $\Gamma^+ = \Gamma \cup \{\cdot\}$.
- The set of movements Π has three elements: *left*, written `<`; *right*, written `>`; and *halt*, written `=`. Then, $\Pi = \{\mathtt{<}, \mathtt{=}, \mathtt{>}\}$.
- The table τ is a total function $\Sigma \times \Gamma^+ \to \Sigma \times \Gamma^+ \times \Pi$, so it can be written as a table of $|\Sigma| \times |\Gamma^+|$ rows and five columns.

¶2 · For example, this is the table for a Turing machine that implements the successor function in base two, $\mathcal{S}_2$. We need two non-*blank* symbols to write base two numerals, so $\Gamma = \{\mathtt{0}, \mathtt{1}\}$, and then $|\Gamma^+| = 3$. We have added a header to help in reading each 5-tuple: C for the current state, R for the read symbol, N for the next state, W for the written symbol, and M for the movement. Two states and three symbols make a six-row table.

C	R	N	W	M
S	·	T	·	<
S	0	S	0	>
S	1	S	1	>
T	·	S	1	=
T	0	T	1	=
T	1	T	0	<

¶3 · To see how this specific Turing machine $\mathcal{S}_2$ works, we can see what happens if we write the string `1101` on the tape, where the numeral `1101` in base 2 is number thirteen. That is, we will write the four symbols `1`, `1`, `0`, and `1` from left to right in four consecutive squares. Then, the initial situation will be as follows.

```
. . . S1 1 0 1 . . .
```

There are two infinities of *blank*s, one to the left and one to the right, that can be omitted, since each one represents an empty string, though we have kept three *blank*s on each side for didactic purposes. The scanned square is the one that has attached to it the current state, which is `S` at the start. Now, `S1` is the index of the third row in the table, so the instruction to do is `S1>`, and therefore the next situation is as follows.

```
. . . 1 S1 0 1 . . .    Thence the machine repeats the instruction.
. . . 1 1 S0 1 . . .    Now the index is S0, so the instruction is S0>.
. . . 1 1 0 S1 . . .    And, again, the instruction for index S1 is S1>.
. . . 1 1 0 1 S. . .    For S ·, it is T · <, so the machine changes the state.
. . . 1 1 0 T1 . . .    And, for index T1, the instruction is T0<.
. . . 1 1 T0 0 . . .    Now, the instruction for T0 is the halting T1=.
. . . 1 1 T1 0 . . .    Therefore, we have reached the final situation.
```

Then the result of the computation, from which all *blank*s are omitted because they represent empty strings, is the string `1110`, which is fourteen in base 2. We can summarize it, thus: $\mathcal{S}_2\langle\mathtt{1101}\rangle \hookrightarrow \mathtt{1110}$.

¶4 · Another way of seeing this Turing machine $\mathcal{S}_2$ is as a Lisp description.

```
(loop
 (set! symbol (read tape))
 (cond
  ((eq? state S)
   (cond
    ((eq? symbol .) (set! state T) (write . tape) (move < tape))
    ((eq? symbol 0) (set! state S) (write 0 tape) (move > tape))
    ((eq? symbol 1) (set! state S) (write 1 tape) (move > tape))))
  ((eq? state T)
   (cond
    ((eq? symbol .) (set! state S) (write 1 tape) (return))
    ((eq? symbol 0) (set! state T) (write 1 tape) (return))
    ((eq? symbol 1) (set! state T) (write 0 tape) (move < tape))))
 ))
```

¶5 · That Lisp description of $\mathcal{S}_2$ can be simplified just by omitting what is not necessary: to set an already set state and to write an already written symbol. And, when the movement is *halt*, the next state is irrelevant. This is a simplified Lisp description of $\mathcal{S}_2$.

```
(loop
 (set! symbol (read tape))
 (cond
  ((eq? state S)
   (cond
    ((eq? symbol .) (set! state T) (move < tape))
    ((eq? symbol 0) (move > tape))
    ((eq? symbol 1) (move > tape))))
  ((eq? state T)
   (cond
    ((eq? symbol .) (write 1 tape) (return))
    ((eq? symbol 0) (write 1 tape) (return))
    ((eq? symbol 1) (write 0 tape) (move < tape))))
 ))
```

¶6 · Note that a Turing machine implementing the successor function in base one, $\mathcal{S}_1$, would be simpler, just two 5-tuples: `S·S1=` and `S1S1>`. You can verify that this Turing machine $\mathcal{S}_1$ will transform the string `111`, for number three in base one, into the string `1111`, which is number four in base one: `S111` $\to$ `1S11` $\to$ `11S1` $\to$ `111S·` $\mapsto$ `111S1`. That is, $\mathcal{S}_1\langle$`111`$\rangle \hookrightarrow$ `1111`.

§7.2 The evolutionary singularity

¶1 · From the problematic point of view, life is the problem of survival, and evolution is its resolver. But evolution works on living beings, each one facing its own individual survival problems and subproblems, which depend on its particular circumstances. So, a way to increase their survival chances is to extend the set of problems that individual living beings solve, thus resulting the evolution of cognition.

¶2 · The problem theory in Casares (P) distinguishes solutions from resolutions. A solution solves the problem, so it is the final target. However, sometimes we do not know any solution to a problem, and how to find a solution to a problem is another problem, its metaproblem. Then, when we do not know any solution to a problem, we have to solve first the metaproblem of the problem. Any solution of a metaproblem is a resolution. Therefore, a resolution is a way to go from a problem to its solutions, and then mathematically a resolution is a function. For instance, in a problem of arithmetical calculation, the solution is a number and the resolution is an algorithm, which is, for example, the algorithm for division when 'by dividing' is the solution to its metaproblem.

It takes 3 eggs to make a cake. How many cakes can you make with 72 eggs?

¶3 · The problem theory presents a set of five resolvers in a series of increasing computing power where each resolver in the series solves at least all the problems that the previous resolver solved. The series of five resolvers is: mechanism, adapter, perceiver, learner, and subject, which can be defined using the distinction between solution and resolution.

- A mechanism implements a solution. It represents the case when we know a solution to the problem, so we can apply it routinely.
- An adapter implements a predicate on solutions, which is mathematically equivalent to a set of solutions. It represents a trial and error on a set of possible solutions.
- A perceiver implements a function on solutions to solutions. It represents the case when we transform a problem into an analogue problem. A special but important case is when the function is a solving resolution, and then it represents the case when, though we do not know a solution, we know a way to get one. In this special case, the analogue problem is the metaproblem.
- A learner implements a predicate on functions or on resolutions, or equivalently it implements a set of functions or of resolutions. It represents a trial and error on a set of possible analogies or resolutions.
- A subject implements a function on functions to functions. When that function is the function implemented by a universal Turing machine, the subject is Turing complete. It represents an analogy on analogies, or a resolution on resolutions.

¶4 · Using this problem theory, this series describes broadly the evolution of cognition: it produced resolvers that were mechanisms at first, then adapters, followed by perceivers, then learners, and finally Turing complete subjects. And, using this theory, evolution is a learner, because evolution is a trial and error on resolvers.

¶5 · Let us now compare problem solving by evolution with problem solving by a Turing complete subject. Evolution is a learning process that has to wait until its prospective hardware resolutions are completely developed adults to find whether they fit or not, while a Turing complete subject is a living individual who can imagine any software resolution in his complete language. Then, comparatively, problem solving by evolution is limited, expensive, and slow, while problem solving by a Turing complete subject is universal, cheap, and quick.

¶6 · An evolutionary singularity is any evolutionary moment when a species surpasses evolution in problem solving. Therefore, whenever evolution produces a Turing complete species there is an evolutionary singularity.

References

Aiello & Wheeler (1995): Leslie C. Aiello, and Peter Wheeler, "The Expensive-Tissue Hypothesis: The Brain and the Digestive System in Human and Primate Evolution"; in *Current Anthropology*, vol. 36, no. 2, pp. 199-221, April 1995, DOI: 10.1086/204350.

Axelrod & Hamilton (1981): Robert Axelrod, and William D. Hamilton, "The Evolution of Cooperation"; in *Science*, New Series, vol. 211, no. 4489, pp. 1390–1396, March 27, 1981, DOI: 10.1126/science.7466396.

Berwick & Chomsky (2016): Robert C. Berwick, and Noam Chomsky, *Why Only Us: Language and Evolution*; The MIT Press, Cambridge MA, 2016, ISBN: 978-0-262-03424-1.

Betker, Fernando & Whalen (1997): Michael R. Betker, John S. Fernando, and Shaun P. Whalen, "The History of the Microprocessor"; in *Bell Labs Technical Journal*, vol. 2, no. 4, pp. 29–56, Autumn 1997, DOI: 10.1002/bltj.2082.

Bingham (1999): Paul M. Bingham, "Human Uniqueness: A General Theory"; in *The Quarterly Review of Biology*, vol. 74, no. 2, pp. 133–169, 1999, DOI: 10.1086/393069.

Casares (C): Ramón Casares, "Proof of Church's Thesis";
DOI: 10.6084/m9.figshare.4955501.

Casares (I): Ramón Casares, "Putnam's Rocks Are Clocks";
DOI: 10.6084/m9.figshare.5450278.

Casares (M): Ramón Casares, "Merge Is Not Recursion";
DOI: 10.6084/m9.figshare.4958822.

Casares (P): Ramón Casares, "Problem Theory";
DOI: 10.6084/m9.figshare.4956353.

Casares (R): Ramón Casares, "On 'On Recursion'";
DOI: 10.6084/m9.figshare.5097691.

Casares (S): Ramón Casares, "Syntax Evolution: Problems and Recursion";
DOI: 10.6084/m9.figshare.4956359.

Casares (U): Ramón Casares, "Universal Grammar is a universal grammar";
DOI: 10.6084/m9.figshare.4956764.

Church (1935): Alonzo Church, "An Unsolvable Problem of Elementary Number Theory"; in *American Journal of Mathematics*, vol. 58, no. 2, pp. 345–363, April 1936, DOI: 10.2307/2371045. Presented to the American Mathematical Society, April 19, 1935.

Darwin (1859): Charles Darwin, "On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life"; John Murray, London, 1859.

Davis (1965): Martin Davis (editor), *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*; Dover, Mineola, New York, 2004, ISBN: 978-0-486-43228-1. Corrected republication of the same title by Raven, Hewlett, New York, 1965.

Dershowitz and Falkovich (2012): Nachum Dershowitz and Evgenia Falkovich, "Honest Universality"; in *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1971, pp. 3340–3348, 18 June 2012, DOI: 10.1098/rsta.2012.0220.

Dollo (1893): Louis Dollo, "Les Lois de l'Evolution"; in *Bulletin de la Société Belge de Géologie, de Paléontologie et d'Hydrologie*, vol. 7, pp. 164–166, 1893.

Einstein (1905): Albert Einstein, "Ist die Trägheit eines Körpers von seinem Energiegehalt abhängig?"; in *Annalen der Physik*, vol. 323, no. 13, pp. 639–641, 1905, DOI: 10.1002/andp.19053231314. English translation by George Barker Jeffery and Wilfrid Perrett: "Does the Inertia of a Body Depend Upon Its Energy Content?"; in *The Principle of Relativity*, London: Methuen and Company, Ltd. (1923).

Fagan (2010): Brian Fagan, *Cro-Magnon: How the Ice Age Gave Birth to the First Modern Humans*; Bloomsbury Press, New York, 2010, ISBN: 978-1-60819-405-6.

Gödel (1930): Kurt Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I"; in *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931, DOI: 10.1007/BF01700692. Received November 17, 1930. English translation in Davis (1965).

Hawks et al. (2000): John Hawks, Keith Hunley, Sang-Hee Lee, and Milford Wolpoff, "Population Bottlenecks and Pleistocene Human Evolution"; in *Molecular Biology and Evolution*, vol. 17, no. 1, pp. 2–22, 2000, DOI: 10.1093/oxfordjournals.molbev.a026233.

Herculano-Houzel & Kaas (2011): Suzana Herculano-Houzel and Jon H. Kaas, "Gorilla and Orangutan Brains Conform to the Primate Cellular Scaling Rules: Implications for Human Evolution"; in *Brain, Behavior and Evolution*, vol. 77, no. 1, pp. 33–44, 2011, DOI: 10.1159/000322729.

Holloway (1985): Ralph L. Holloway, "The Poor Brain of Homo sapiens neanderthalensis: See What You Please..."; in *Ancestors: The Hard Evidence*, pp. 319–324, edited by E. Delson, New York: Alan R. Liss, 1985, ISBN: 0-47184376-8.

Huizinga (1938): Johan Huizinga, "Homo Ludens: A Study of the Play-Element in Culture"; London: Routledge & Kegan Paul, London, 1949, 1980, ISBN: 0-7100-0578-4.

Kleene (1935): Stephen Kleene, "General Recursive Functions of Natural Numbers"; in *Mathmatische Annalen*, vol. 112, pp. 727–742, 1936, DOI: 10.1007/BF01565439. Presented to the American Mathematical Society, September 1935.

Kleene (1936): Stephen Kleene, "λ-Definability and Recursiveness"; in *Duke Mathematical Journal*, vol. 2, pp. 340–353, 1936, DOI: 10.1215/s0012-7094-36-00227-2.

Kruska (1988): D. Kruska, "Effects of Domestication on Brain Structure and Behavior in Mammals"; in *Human Evolution*, vol. 3, no. 6, pp. 473–485, 1988, DOI: 10.1007/bf02436333.

Kruska (2005): Dieter C.T. Kruska, "On the Evolutionary Significance of Encephalization in Some Eutherian Mammals: Effects of Adaptive Radiation, Domestication, and Feralization"; in *Brain, Behavior and Evolution*, vol. 65, no. 2, pp. 73–108, 2005, DOI: 10.1159/000082979.

McCarthy (1960): John McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I"; in *Communications of the ACM*, vol. 3, no. 4, pp. 184–195, April 1960, DOI: 10.1145/367177.367199.

McCarthy et al. (1962): John McCarthy, Paul Abrahams, Daniel Edwards, Timothy Hart, and Michael Levin, *LISP 1.5 Programmer's Manual*; The MIT Press, Cambridge, Massachusetts, 1962, ISBN: 978-0-262-13011-0.

Mealy (1955): George H. Mealy, "A Method for Synthesizing Sequential Circuits"; in *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, September 1955, DOI: 10.1002/j.1538-7305.1955.tb03788.x.

Neary and Woods (2006): Turlough Neary and Damien Woods, "Small Fast Universal Turing Machines"; in *Theoretical Computer Science*, vol. 362, no. 1–3, pp. 171–195, 11 October 2006, DOI: 10.1016/j.tcs.2006.06.002.

Patterson et al. (2006): N. Patterson, D.J. Richter, S. Gnerre, E.S. Lander, and D. Reich, "Genetic evidence for complex speciation of humans and chimpanzees"; in *Nature*, vol. 441, pp. 1103–1108, 2006, DOI: 10.1038/nature04789.

Pavlov (1927): Ivan Pavlov, *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*, translated by G.V. Anrep; Oxford University Press, London, 1927. Reprinted by Dover, Minneola, NY, 1960, 2003, ISBN: 0-486-43093-6.

Post (1936): Emil Post, "Finite Combinatory Processes — Formulation 1"; in *The Journal of Symbolic Logic*, Volume 1, Number 3, pp. 103–105, September 1936. Received October 7, 1936. DOI: 10.2307/2269031.

Post (1944): Emil L. Post, "Recursively Enumerable Sets of Positive Integers and their Decision Problems"; in *Bulletin of the American Mathematical Society*, vol. 50, no. 5, pp. 284–316, 1944, DOI: 10.1090/s0002-9904-1944-08111-1.

Putnam (1988): Hilary Putnam, *Representation and Reality*; The MIT Press, Cambridge, MA, 1988, ISBN: 978-0-262-66074-7.

Searle (1980): John R. Searle, "Minds, Brains, and Programs"; in *The Behavioral and Brain Sciences*, vol. 3, no. 3, pp. 417–424, September 1980, DOI: 10.1017/S0140525X00005756.

Searle (1992): John R. Searle, *The Rediscovery of the Mind*; The MIT Press, Cambridge, MA, 1992, ISBN: 978-0-262-69154-3.

Shannon (1948): C. E. Shannon, "A Mathematical Theory of Communication"; in *Bell System Technical Journal*, vol. 27, no. 3 & no. 4, pp. 379–423 & 623–656, DOI: 10.1002/j.1538-7305.1948.tb01338.x & DOI: 10.1002/j.1538-7305.1948.tb00917.x.

Shannon (1956): Claude E. Shannon, "A Universal Turing Machine with Two Internal States"; in *Automata Studies, Annals of Mathematics Studies*, vol. 34, pp. 157–165, 1956, DOI: 10.1515/9781400882618-007.

Tomasello (2008): Michael Tomasello, *Origins of Human Communication*; The MIT Press, Cambridge, Massachusetts, 2008, ISBN: 978-0-262-51520-7.

Tomasello (2009): Michael Tomasello. "Universal Grammar Is Dead"; in *Behavioral and Brain Sciences*, vol. 32, no. 5, pp. 470–471, 2009, DOI: 10.1017/S0140525X09990744.

Turing (1936): Alan Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem"; in *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937, DOI: 10.1112/plms/s2-42.1.230. Received 28 May, 1936. Read 12 November, 1936.

Turing (1937): Alan Turing, "Computability and λ-Definability"; in *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, December 1937, DOI: 10.2307/2268280.

Watumull et al. (2014): Jeffrey Watumull, Marc D. Hauser, Ian G. Roberts, and Norbert Hornstein, "On Recursion"; in *Frontiers in Psychology*, vol. 4, art. 1017, January 2014, DOI: 10.3389/fpsyg.2013.01017.