# Gödel Incompleteness and Turing Completeness

**Ramón Casares**

ORCID: 0000-0003-4973-3128

*Following Post program, we will propose a linguistic and empirical interpretation of Gödel's incompleteness theorem and related ones on unsolvability by Church and Turing. All these theorems use the diagonal argument by Cantor in order to find limitations in finitary systems, as human language, which can make "infinite use of finite means". The linguistic version of the incompleteness theorem says that every Turing complete language is Gödel incomplete. We conclude that the incompleteness and unsolvability theorems find limitations in our finitary tool, which is our complete language.*

*Keywords:* *Gödel incompleteness, Turing completeness, law of Post, Cantor's diagonal argument*

## §1 Introduction

¶1 · Following Post (1936) program, we will argue in favor of a linguistic and empirical interpretation of Gödel's (1930) incompleteness theorem and related ones on unsolvability by Church (1935) and Turing (1936). All these theorems use the diagonal argument by Cantor (1891) in order to find limitations in finitary systems, as human language, which can make "infinite use of finite means". The linguistic version of the incompleteness theorem says, see §6¶8, that *every Turing complete language is Gödel incomplete.*

¶2 · In section §2, we explain Cantor's diagonal argument. Next, in §3, we sketch Gödel's incompleteness theorem, which uses the diagonal argument on an *ad hoc* language. Then, in §4 and §5, we present Turing computing: Turing completeness and complete languages. In §6, we show that the halting problem generalizes Gödel's theorem and, using the diagonal argument on a complete language, that it is unsolvable by computing. Then, following Post program, in §7 and §8, we posit as a refutable law of nature that human language is just complete, by which the incompleteness and unsolvability theorems become human limitations. Next, in §9, we defend, in Kant's terminology, that language is a condition of all possible theory, and, in §10, that language can "make infinite use of finite means", in Humboldt's words. Finally, in §11, after noticing that the diagonal argument finds limitations in finitary systems, we conclude that the incompleteness and unsolvability theorems find limitations in our finitary tool, which is our complete language.

This is DOI: 10.6084/m9.figshare.25434994.v2, version 20240424.

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

## §2 Cantor

¶1 · Let us start from the beginning. In a four pages paper, Cantor (1891) presented a new and simpler proof showing that the cardinality of the real numbers $|\mathbb{R}|$ is greater than the cardinality of the natural numbers $|\mathbb{N}|$, that is, $|\mathbb{R}| > |\mathbb{N}|$.

¶2 · The proof shows that the powerset (the set of all the subsets) of the natural numbers cannot be enumerated. First, we should see that every subset $R$ is defined by a predicate on the natural numbers $R(n)$ that is TRUE if the natural number $n$ belongs to the subset, and FALSE if it does not. Fully expressed:

$$\forall n \in \mathbb{N} \begin{cases} R(n) = \text{TRUE} & \text{if } n \in R \,, \\ R(n) = \text{FALSE} & \text{if } n \notin R \,. \end{cases}$$

Or, in fewer words,

$$n \in R \equiv R(n) \,.$$

¶3 · For the sake of the argument, let us assume that all the predicates defining the subsets of the natural numbers can be enumerated. In that case, we could denote $R_p$ the predicate number $p$, and we could compose the following matrix of TRUE and FALSE values.

$$\begin{array}{cccccc} R_0(0) & R_0(1) & R_0(2) & \dots & R_0(n) & \dots \\ R_1(0) & R_1(1) & R_1(2) & \dots & R_1(n) & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \\ R_p(0) & R_p(1) & R_p(2) & \dots & R_p(n) & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \end{array}$$

¶4 · Now, let us define a subset $K$ this way, where the bar above denotes negation:

$$n \in K \equiv \overline{R_n(n)} \,.$$

It is easy to see that $K \neq R_0$, because, if $0 \in R_0$ then $R_0(0) = \text{TRUE}$, so $\overline{R_0(0)} = \text{FALSE}$, and then $0 \notin K$, and conversely, if $0 \notin R_0$ then $0 \in K$. Similarly for 1, and then $K \neq R_1$, and for 2, so $K \neq R_2$, and so on for every natural number. Therefore, $\forall p \in \mathbb{N} : K \neq R_p$, showing that the assumption was wrong.

¶5 · The proof shows that the powerset of the natural numbers cannot be enumerated, in Cantor's notation $\aleph_0 < 2^{\aleph_0}$. This is enough for us here, but Cantor was interested in showing that $|\mathbb{N}| < |\mathbb{R}|$, which follows immediately from $|\mathbb{N}| = \aleph_0$ and $|\mathbb{R}| = 2^{\aleph_0}$. The proof is named Cantor's *diagonal argument* because the elements chosen to be negated are those in the matrix diagonal, $R_n(n)$, which are the most easily denoted, though the proof works just by choosing systematically a different column for each row. Below, in §6¶1, we will learn from Gödel that choosing to negate the diagonal instantiates the liar paradox; other selections will implement other epistemological antinomies.

*God made the counting numbers;*
*all else is the work of man*
LEOPOLD KRONECKER

## §3 Gödel

¶1 · In order to explain the incompleteness theorem by Gödel (1930), where it is Theorem VI but sometimes referred to as the first one, we will start showing the following lemma: *in every language that is expressive enough to mean 'this sentence is false' there is a paradox*. It is easy to see that the lemma is true because the sentence 'this sentence is false' is a paradox; in fact it is the prototypical liar paradox. Therefore, the hard part of the incompleteness theorem is to prove that *the language required to express arithmetic has to be expressive enough to mean 'this sentence is false'*. Let us try it!

¶2 · The language required to express arithmetic has to be able to express that 'the successor of zero is one', which happens to be considered TRUE, that 'one plus one is one', considered FALSE, that 'thirteen is prime' and also that 'thirteen is not prime', and an infinity of other propositions. I have written 'an infinity of other propositions', perhaps too lightly, but am I right? The answer is 'yes', because there is an infinite enumerable number of TRUE propositions as 'the successor of zero is one', each one referring to a different natural number, as 'the successor of one is two', 'the successor of two is three', and so on and on. This means that the language required to express arithmetic has to be infinite enumerable, at least.

¶3 · Although the infinite enumerable sets are the infinite sets with the lowest cardinal number, $\aleph_0$, it is still true that all of them contain proper subsets that are equicardinal with the whole set. This property is required for full naming, also known as full referring, that is, to be able to assign a different name to every linguistic object. For example, Gödel (1930) was able to assign a unique natural number, its Gödel number which worked as its name, to each and every arithmetic object, including the natural numbers themselves! Of course, self referring, included in full referring, is used in paradoxes as 'this sentence is false'.

¶4 · Above, we were using English statements to express arithmetic propositions, but Gödel, instead of using German, designed a more precise language. All the concepts used in the proof are implemented in the formal system. We follow Gödel's (1930) sketch of the proof, in page 175, as translated by Davis (1965) in pages 7 and 8. Gödel's subset $K$ of natural numbers is defined

$$n \in K \equiv \overline{\text{Bew}}[R_n(n)] \,,$$

where $\text{Bew}[x]$ means '$x$ is a provable formula', and where $R_n(n)$ is Cantor's diagonal formalized (this last word is decisive!). Since $K$ is a subset of $\mathbb{N}$, then its predicate has to be some definite enumerated predicate $R_q$, so $K$ can also be defined

$$n \in K \equiv R_q(n) \,.$$

We now show that the proposition $R_q(q)$ is undecidable in the formal system. If we assume that $R_q(q)$ is provable, then it would be TRUE, and then, following the second definition, $q \in K$, which, following the first definition, means that $\overline{\text{Bew}}[R_q(q)]$, contradicting the assumption. On the other hand, if the negation of $R_q(q)$ were provable, then $q \notin K$ would hold following the second definition, and then $\text{Bew}[R_q(q)]$ would be TRUE following the first one, but now both $R_q(q)$ together with its negation would be provable, which is again impossible.

¶5 · So Gödel is setting out of sight the following Cantor's matrix of provabilities for every formalized predicate, which is TRUE if $R_p(n)$ is provable, so $R_p(n)$ is TRUE, and FALSE if the negation of $R_p(n)$ is provable.

$$\begin{array}{ccccccc}
\text{Bew}[R_0(0)] & \text{Bew}[R_0(1)] & \text{Bew}[R_0(2)] & \dots & \text{Bew}[R_0(n)] & \dots \\
\text{Bew}[R_1(0)] & \text{Bew}[R_1(1)] & \text{Bew}[R_1(2)] & \dots & \text{Bew}[R_1(n)] & \dots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots \\
\text{Bew}[R_p(0)] & \text{Bew}[R_p(1)] & \text{Bew}[R_p(2)] & \dots & \text{Bew}[R_p(n)] & \dots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots
\end{array}$$

The negated diagonal of this matrix is $\overline{\text{Bew}}[R_n(n)]$, on which he defines subset $K$ with its corresponding predicate $R_q$ and undecidable proposition $R_q(q)$, which Gödel writes $[R(q);q]$, which expresses the liar paradox in the formal system, see §6¶1.

¶6 · As you can see, Gödel (1930) is not as simple as Cantor (1891) and, given Cantor's proof, some doubts regarding whether $R_q$ is effectively in the enumeration or not could remain that the full proof conceals behind its many details. Gödel's proof is not as simple because it implements a comprehensive and logical language specifically designed to express arithmetic. However, we will not go further into its technicalities here, because Turing (1936) turned things much easier, and then much easier to understand, I hope.

## §4 Turing

¶1 · Instead of using symbols specifically designed to refer to arithmetic concepts, as done by Gödel (1930), Turing (1936) uses a non-empty and finite set of arbitrary symbols. Thus, Turing's way is more general than Gödel's one and, even more important, less prone to induce us to go from the symbol to its intended meaning. All Turing requires is one or more symbols that can be strung, that is, that can be composed into unidimensional and finite structures, which are known as *string*s.

¶2 · And again, where Gödel implements the rules of inference specific of arithmetic and logic to transform arithmetic formulas, Turing uses generic finite-state machines, also known as finite automata, to transform the generic strings of symbols. Note that finite-state machines were not formally studied until much later than Turing's 1936 paper, in the nineteen-fifties by Mealy (1955), Moore (1956), and others.

¶3 · How Turing machines work is not hard to understand, and you can find many other places where it is explained, for example in Casares (T). For us here, it is enough to know that a Turing machine takes a finite string of symbols written on its tape as input and, commanded by the finite-state machine, it halts after a finite number of computing steps with another finite string written on its tape, which is its output, or it keeps computing without halting. We will denote $\mathcal{P}\langle\mathfrak{d}\rangle$ the output string of Turing machine $\mathcal{P}$ when the string $\mathfrak{d}$ was used as input; if the machine does not halt, we will write $\mathcal{P}\langle\mathfrak{d}\rangle = \infty$.

¶4 · Now, the deepest and more difficult to understand concept is *Turing completeness*, that is, that some Turing machines can compute whatever any Turing machine can compute. Those Turing machines that can emulate any Turing machine are called *universal Turing machines*. Turing (1936) himself constructs a universal machine, but I would very much recommend an instructive text book, as Abelson & Sussman (1985), to get the full details of this counterintuitive concept.

¶5 · In order to imitate any possible Turing machine $\mathcal{P}$, a universal Turing machine, which we will denote $\mathcal{U}$, needs a complete description of the Turing machine to imitate as input, that is, as part of what is written on the tape when it starts. We will call this complete description the *program*, and we will denote $\mathfrak{p}$ the program for $\mathcal{P}$. So the *equation of Turing completeness*, where | is an end of program symbol, is:

$$\exists\mathcal{U}, \forall\mathcal{P}, \forall\mathfrak{d} : \overleftarrow{\mathcal{U}\langle\, \mathfrak{p}|\vec{\mathfrak{d}}\,\rangle} = \mathcal{P}\langle\mathfrak{d}\rangle\ .$$

We say that $\mathcal{U}$ is a *full-programmable* computer because it can compute whatever any computer $\mathcal{P}$ can compute. Let us now examine the equation closely.

¶6 · Today, the existence of universal computers, $\exists\mathcal{U}$, is a common experience, since all general-purpose computers, including our phones, are *Turing complete*, or *universal*, or *full-programmable*; these last three phrases are synonymous. For the equation to be true, the next task is to show that for any Turing machine, $\forall\mathcal{P}$, we can find a string of symbols $\mathfrak{p}$ (symbols of $\mathcal{U}$) that describes the Turing machine $\mathcal{P}$ completely. This is not too difficult; just linearize the table defining $\mathcal{P}$'s finite-state machine, and code the states and symbols of $\mathcal{P}$ by using strings composed of symbols of $\mathcal{U}$; we will denote this $\mathfrak{p} = \vec{\mathcal{P}}$. For example, Turing (1936) in his universal machine uses a semicolon (`;`) as end of row symbol to linearize the table, and codes $\mathcal{P}$ state number $i$ as one symbol `D` followed by $i$ symbols `A`, and $\mathcal{P}$ symbol number $i$ as one symbol `D` followed by $i$ symbols `C`. Other codings are possible; the only condition for a coding is that it sets a reversible mapping from the states and symbols of $\mathcal{P}$ to some strings of symbols of $\mathcal{U}$. This determines the *syntax* of the language used by each specific $\mathcal{U}$ to code any possible Turing machine $\mathcal{P}$ as its corresponding program, the string $\mathfrak{p} = \vec{\mathcal{P}}$, and also to code any string $\mathfrak{d}$ of symbols of $\mathcal{P}$, denoted $\vec{\mathfrak{d}}$; a left pointing arrow on top denotes the corresponding decoding, $\overleftarrow{\vec{\mathfrak{d}}} = \mathfrak{d}$· The remaining task is the difficult one: to implement the *semantics* of Turing machines in the hardware of $\mathcal{U}$, that is, in its finite-state machine, in such a way that the equation of Turing completeness will be satisfied, ever (including $\infty$). Once achieved, the resulting language is a Turing complete language, or a *complete language* for short. Therefore, *in a complete language, every Turing machine can be meaningfully expressed.*

## §5 Translation

¶1 · As the syntax can be defined in different ways, the program $\mathfrak{p}'$ for Turing machine $\mathcal{P}$ in one universal Turing machine $\mathcal{U}'$ will differ from the program $\mathfrak{p}''$ for the same Turing machine $\mathcal{P}$ in another universal Turing machine $\mathcal{U}''$ that uses a different syntax, that is,

$$\mathfrak{p}' \neq \mathfrak{p}'' \text{ though } \overleftarrow{\mathcal{U}'\langle\mathfrak{p}'|\vec{\mathfrak{d}}'\rangle}' = \mathcal{P}\langle\mathfrak{d}\rangle = \overleftarrow{\mathcal{U}''\langle\mathfrak{p}''|\vec{\mathfrak{d}}''\rangle}''\ .$$

We will call this last double equation the *translation equation* because it explains how to translate complete languages, in this case from (or to) the complete language $\mathcal{L}'$ implemented by $\mathcal{U}'$ to (or from) the complete language $\mathcal{L}''$ implemented by $\mathcal{U}''$, since though their syntaxes differ, $\mathfrak{p}' \neq \mathfrak{p}''$, their meanings are the same, $\mathcal{P}$. The equations of Turing completeness and translation imply that *all universal Turing machines are equal in calculating capability*, since they differ only on the encodings used.

¶2 · At this point, we can abstract encodings, that is, codings and decodings, away, since coding (or decoding) is a trivial transformation that uses a finite mapping, and it is easy to determine from context whether coding (or decoding) is needed or not. Then, we define the *algorithmic equivalence relation* thus: two programs $\mathfrak{p}'$ and $\mathfrak{p}''$ are algorithmically equivalent, denoted $\mathfrak{p}' \sim \mathfrak{p}''$, iff their meanings are the same; for instance when they are translations of the same Turing machine $\mathcal{P}$ to two complete languages, $\mathcal{L}'$ and $\mathcal{L}''$:

$$\mathfrak{p}' \sim \mathfrak{p}'' \equiv \left( \exists \mathcal{P}, \forall \mathfrak{d} : \overleftarrow{\mathcal{U}' \langle \mathfrak{p}' | \vec{\mathfrak{d}}' \rangle}' = \mathcal{P}\langle \mathfrak{d} \rangle = \overleftarrow{\mathcal{U}'' \langle \mathfrak{p}'' | \vec{\mathfrak{d}}'' \rangle}'' \right).$$

¶3 · The corresponding equivalence classes are called algorithms, so algorithm $\pi$ is the equivalence class of program $\mathfrak{p}$, that is, $\pi = [\mathfrak{p}] = \{ \mathfrak{x} : \mathfrak{x} \sim \mathfrak{p} \}$. We are using uppercase calligraphic letters to denote Turing machines, as $\mathcal{P}$ and $\mathcal{U}$, German lowercase letters for strings in the tapes of Turing machines, as $\mathfrak{p}$ and $\mathfrak{d}$, or $\vec{\mathfrak{d}}$ if coded, but typewriter characters for individual symbols, as `A` and `w`, and we will use Greek lowercase letters for algorithms and information, as $\pi$ and $\delta$, where information is data after abstracting away its encoding. And, after abstracting encodings away, the equation of Turing completeness is cleaner; the capital upsilon $\Upsilon$ represents the abstract universal Turing machine:

$$\Upsilon \langle \pi | \delta \rangle \cong \mathcal{P} \langle \mathfrak{d} \rangle .$$

¶4 · While Gödel (1930) mimics semantics into syntax from the beginning, making the distinction between both more difficult to grasp, in Turing's (1936) approach, meanings appear only when completeness appears. That is, everything is syntax, except when implementing a language to fully express computing, because by then implementing the semantics of computing is required. However, after abstracting encodings away, it is also possible to confuse concepts in computing. That is, given the coding-decoding bijection between Turing machines and programs in a complete language, $\{\mathcal{P}\} \leftrightarrow \{\mathfrak{p}\}$, it is nearly natural to use $\mathfrak{p}$ for $\mathcal{P}$, or $\mathcal{P}$ for $\mathfrak{p}$, and after abstracting encodings away, $\pi = [\mathfrak{p}]$, it is only a minor inconvenience to use $\pi$ for $\mathfrak{p}$, or $\mathfrak{p}$ for $\pi$. And the same ambiguity can be applied to data $\mathfrak{d}$, coded data $\vec{\mathfrak{d}}$, and information $\delta$. But it is much better not to confuse these three levels:

- Semantics: hardware ($\mathcal{P}$) and data ($\mathfrak{d}$).
- Syntax: software ($\mathfrak{p} = \vec{\mathcal{P}}$) and coded data ($\vec{\mathfrak{d}}$).
- Pragmatics: algorithms ($\pi$) and information ($\delta$).

¶5 · By abstracting encodings away we are ignoring syntax. We could do it, but it would be dangerous because, in computing, syntax is prior to both semantics and pragmatics. In the beginning everything is syntax, because a Turing machine is a finite-state machine applying its syntactic rules to generic strings of symbols. It is later, when implementing a universal Turing machine, that a complete language is required that gives meaning to the whole of computing; so semantics is required to implement complete languages. And finally, if we abstract encodings away, we get language independent knowledge; now we can resolve problems algorithmically, meaning that their solutions can be coded in different complete languages and implemented in different hardware devices. However, to pragmatically resolve a problem, the algorithm has to be instantiated, that is, coded on a specific universal Turing machine, or implemented directly on a specific piece of hardware. All things considered, language independent knowledge could be misleading, since it promises more than it provides; use it with caution!

## §6 Proof

¶1 · So we are seeing that Gödel (1930) is generalized by Turing (1936). Then we need to ascertain what generalizes Gödel's incompleteness theorem in Turing's computing. As Gödel (1930) writes in page 175, translated by Davis (1965) in page 9, "there is also a close relationship[14] with the Liar paradox, for the undecidable proposition $[R(q);q]$ says $[\dots]$ that $[R(q);q]$ is not provable." Note 14 says: "Every epistemological antinomy can be used for a similar proof of undecidability." In any case, he uses the canonical liar paradox, 'this sentence is false', that has not a definite meaning because it can neither be decided true nor false; if it is true what it says, then it is false, but if it is false what it says, then it is true, thus closing an infinite loop. In Turing computing, the only computations that result undecided are those that do not halt.

¶2 · Therefore, the generalization of Gödel's incompleteness theorem is the theorem showing that the halting problem is unsolvable by a Turing machine. Though, in fact, the halting problem was defined later by Davis (1958), Turing (1936) had already shown that it is unsolvable, by resolving the circularity problem, see Petzold (2008) page 179.

¶3 · Without loss of generality, we will use the complete language $\mathcal{L}$ implemented by the universal Turing machine $\mathcal{U}$ for the proof. $\mathcal{L}$ has a finite set of symbols, and therefore the set of its finite strings $\{\,\mathfrak{d}\,\}$ is enumerable, using for example a lexicographic order. To determine whether a string $\mathfrak{d}$ of $\mathcal{U}$ is a coded string $\vec{\mathfrak{d}}$ or not is easily decidable, as its syntax is straight, just a simple mapping, and then the set of coded strings $\{\,\vec{\mathfrak{d}}\,\}$ is enumerable. Then we will refer to coded string number $d$ as $\vec{\mathfrak{d}}_d$. To determine whether a string codes a Turing machine or not is also easily decidable, as its syntax is simple, a linearized table of simply coded states and symbols. The conclusion, which was perhaps doubtful in Gödel's proof, see §3¶6, it is easy to see in computing: the set of programs $\{\,\mathfrak{p}\,\}$ is enumerable. Then we will refer to program number $p$ as $\mathfrak{p}_p$.

¶4 · We will set a Cantor's diagonal argument to show that there is not any Turing machine $\mathcal{H}$ that takes any arbitrary pair of program $\mathfrak{p}$ and coded data $\vec{\mathfrak{d}}$ as input, and every time it outputs, in a finite number of computing steps, a string expressing whether $\mathcal{U}\langle\,\mathfrak{p}|\vec{\mathfrak{d}}\,\rangle$ will halt or run indefinitely, say the one symbol string `Y` if it will halt, and `N` otherwise.

$$\nexists\mathcal{H},\forall\mathfrak{p},\forall\vec{\mathfrak{d}}\begin{cases}\mathcal{H}\langle\,\mathfrak{p}|\vec{\mathfrak{d}}\,\rangle=\mathtt{Y} & \text{if } \mathcal{U}\langle\,\mathfrak{p}|\vec{\mathfrak{d}}\,\rangle \text{ halts}\\ \mathcal{H}\langle\,\mathfrak{p}|\vec{\mathfrak{d}}\,\rangle=\mathtt{N} & \text{if } \mathcal{U}\langle\,\mathfrak{p}|\vec{\mathfrak{d}}\,\rangle \text{ does not halt}\end{cases}$$

¶5 · Now, for the sake of the argument, let us assume that $\mathcal{H}$ exists. Then the following matrix of `Y` and `N` string values could be computed (without got stuck in a non-halting computation).

$$\begin{array}{ccccccc}
\mathcal{H}\langle\mathfrak{p}_0|\vec{\mathfrak{d}_0}\rangle & \mathcal{H}\langle\mathfrak{p}_0|\vec{\mathfrak{d}_1}\rangle & \mathcal{H}\langle\mathfrak{p}_0|\vec{\mathfrak{d}_2}\rangle & \dots & \mathcal{H}\langle\mathfrak{p}_0|\vec{\mathfrak{d}_d}\rangle & \dots\\
\mathcal{H}\langle\mathfrak{p}_1|\vec{\mathfrak{d}_0}\rangle & \mathcal{H}\langle\mathfrak{p}_1|\vec{\mathfrak{d}_1}\rangle & \mathcal{H}\langle\mathfrak{p}_1|\vec{\mathfrak{d}_2}\rangle & \dots & \mathcal{H}\langle\mathfrak{p}_1|\vec{\mathfrak{d}_d}\rangle & \dots\\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots\\
\mathcal{H}\langle\mathfrak{p}_p|\vec{\mathfrak{d}_0}\rangle & \mathcal{H}\langle\mathfrak{p}_p|\vec{\mathfrak{d}_1}\rangle & \mathcal{H}\langle\mathfrak{p}_p|\vec{\mathfrak{d}_2}\rangle & \dots & \mathcal{H}\langle\mathfrak{p}_p|\vec{\mathfrak{d}_d}\rangle & \dots\\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots
\end{array}$$

However, the row for program $\mathfrak{q}$, which negates the diagonal, would not be in the matrix.

$$\mathfrak{q}:\forall n\in\mathbb{N}\begin{cases}\mathcal{U}\langle\mathfrak{q}|\vec{\mathfrak{d}_n}\rangle=\mathtt{Y} & \text{if } \mathcal{H}\langle\mathfrak{p}_n|\vec{\mathfrak{d}_n}\rangle=\mathtt{N}\\ \mathcal{U}\langle\mathfrak{q}|\vec{\mathfrak{d}_n}\rangle=\infty & \text{if } \mathcal{H}\langle\mathfrak{p}_n|\vec{\mathfrak{d}_n}\rangle=\mathtt{Y}\end{cases}$$

Program $\mathfrak{q}$ would exist if the program for $\mathcal{H}$, denoted $\mathfrak{h}$, existed, which would be the case if $\mathcal{H}$ existed as assumed. Therefore, the assumption was false.

¶6 · Then, in the complete language $\mathcal{L}$ of the universal Turing machine $\mathcal{U}$, a program $\mathfrak{h}$ that solves the halting problem is inexpressible. Of course, taking advantage of the translation equation, this theorem holds for every complete language, which then can be formulated this way: *in every complete language, there is an inexpressible program.*

¶7 · In computing, to say that there are undecidable computations is too trivial to be of any interest, since it just means that some computations do not halt, as for example `while true { relax }` or `liar() = return( not liar() )`. However, it is not only that in any complete language there are computations that do not halt, but also that there is not any computable way of avoiding them definitively. This parallels Gödel's conclusion that undecidable propositions cannot be avoided by adding them as axioms; you just need to use Cantor's diagonal on the new formal system to find a new undecidable proposition.

¶8 · From Turing's conclusion that there are problems that cannot be solved by computing, it follows that, in every complete language, there are expressible problems the solutions of which are not expressible, so we can say that in every complete language there are concepts that can be defined, and named, but not fully expressed. Then a funnier way to state Turing's conclusion, which is closer to popular formulations of Gödel's incompleteness theorem, is:

*every complete language is not complete.*

This statement uses two different meanings of the word 'complete': the complete language is (Turing) complete because every Turing machine can be meaningfully expressed in it, but it is not (Gödel) complete because there are definable tasks that no Turing machine can perform and, consequently, they cannot be expressed in it.

## §7 Church

¶1 · Some of my readers could suspect that I am pretending to pass as new that Turing generalizes Gödel. It is not my intention. In fact, this is known from the very beginning, since Turing (1936) himself, in page 259, shows that Gödel's incompleteness theorem is a consequence of his unsolvability theorem. This is his argument, where *Entscheidungsproblem* is the German word for 'decision problem':

> *If the negation of what Gödel [(1930)] has shown had been proved, i.e. if, for each $\mathfrak{A}$, either $\mathfrak{A}$ or $-\mathfrak{A}$ is provable [in the functional calculus $\mathsf{K}$], then we should have an immediate solution of the Entscheidungsproblem. For we can invent a [Turing] machine $\mathcal{K}$ which will prove consecutively all provable formulae. Sooner or later $\mathcal{K}$ will reach either $\mathfrak{A}$ or $-\mathfrak{A}$. If it reaches $\mathfrak{A}$, then we know that $\mathfrak{A}$ is provable. If it reaches $-\mathfrak{A}$, then, since $\mathsf{K}$ is consistent [...], we know that $\mathfrak{A}$ is not provable.*

In other words, if Gödel's incompleteness theorem were not the case, $\overline{\mathfrak{G}}$, then Turing's unsolvability of the decision problem would not be the case, $\overline{\mathfrak{T}}$, denoted $\overline{\mathfrak{G}} \to \overline{\mathfrak{T}}$, which is equivalent by contraposition to $\mathfrak{T} \to \mathfrak{G}$, which means that Turing's implies Gödel's, or in reverse, that Gödel's is a logical consequence of Turing's.

¶2 · As these results on decidability and on solvability are all negative, a question arises: Could it be that they can be decided and solved by devices that are more capable than universal Turing machines? The accepted answer is *Church's thesis*, which asserts that

*there are not more capable calculating devices than universal Turing machines.* Before going on, please note that, in linguistic terms, the question is a bit silly, since it is asking for a language more expressive than a complete language but in which the sentence 'this sentence is false' is not expressible, a sentence that is expressible in every complete language. Linguistically, the impossibility is apparent.

¶3 · The interesting story, or history, around Church's thesis is detailed by Davis (1982), who explains why Gödel preferred Turing machines over his own recursive functions and over Church's λ-calculus in order to fix Church's thesis. As we have written above, the advantage of Turing computable functions, which are those from strings to strings implemented by Turing machines, is that they are much more generic and simpler than Gödel's recursive functions, and the same applies to Church's λ-definable functions. For example, in the case of the recursive functions, in order to achieve a capability equivalent to Turing completeness, it was necessary to add a conditional incremental loop to the conditional decremental loop of primitive recursion, see Kleene (1952) Chapter XI, an addition which we could describe as a hack, and even then Gödel "was [...] not at all convinced that [his] concept of recursion comprises all possible recursions", as cited by Davis (1982) in page 8. This explains why we have used universal Turing machines when we presented Church's thesis, while Church (1935) himself used recursive functions and λ-calculus instead, and it also explains why, though he resolved the Entscheidungsproblem as unsolvable before Turing (1936), we prefer the cleaner proof by Turing.

¶4 · According to Kleene (1952), from page 319 on, the arguments supporting Church's thesis as the accepted answer are of four types: heuristic evidence, equivalence of diverse formulations, Turing's concept of a computing machine, and symbolic logics and symbolic algorithms. As an example of the second type, it is really convincing that, even being so different, computing, recursive functions, and λ-calculus are all capable of universality. The computing version of universality is Turing completeness, that is, that computing can express computing completely, as we saw above. In the case of recursive functions, it is Kleene's (1935a) normal form, see Kleene (1952) Theorem IX in page 288, where a recursive function can express any recursive function. And, for Church's λ-calculus, it is that an evaluator of λ-expressions can be defined as a λ-defined function, resulting that the evaluator is a λ-defined function able to express any λ-defined function; this is done (in Lisp) by Abelson & Sussman (1985). Universality, or full-self-expressibility, manifests itself in all three formulations because they are equivalent, as shown by Kleene (1935b) and Turing (1937).

¶5 · To me, that all formulations of the concept of computing point to the same calculating limit, which is universality, suggests that the limit has an empirical meaning. For suppose these two possibilities:

- Tomorrow someone devises a procedure to perform calculations that are beyond the capability of a universal Turing machine.
- Tomorrow some machine is found that performs calculations that are beyond the capability of a universal Turing machine.

In either case, Church's thesis would be wrong. Therefore, Church's thesis is uncertain because it depends on what it might occur tomorrow, showing that it is empirically refutable.

## §8 Post

¶1 · In other words, I am with Post (1936) when he concludes, in page 105:

> *Only so, [that is, only if Church's thesis is a natural law stating the limitations of the mathematicizing power of our species Homo sapiens], can Gödel's theorem concerning the incompleteness of symbolic logics of a certain general type and Church's results on the recursive unsolvability of certain problems be transformed into conclusions concerning all symbolic logics and all methods of solvability.*

That *natural law*, deservedly named the *law of Post*, can be stated as follows:

*in calculating capability, we are just Turing complete.*

The corresponding linguistic version of the law of Post is:

*in expressive capability, human language is just Turing complete.*

¶2 · Church's thesis is a consequence of the law of Post because, as shown in Casares (C), where the law of Post is introduced, if we are *just* Turing complete, then Church's thesis is true. As any natural law, the law of Post is empirically refutable and then it is in need of continual verification. It can be refuted by empirical evidence negating it, or by empirical evidence negating any of its consequences, for example by evidence negating Church's thesis. Therefore, what is supporting both the law of Post and Church's thesis is the lack of empirical evidence on the contrary, for the time being. Nothing more, nothing less.

¶3 · For example, hypercomputers would refute Church's thesis. According to Shagrir & Pitowsky (2003): "A hypercomputer is a physical or an abstract system that computes functions that cannot be computed by a universal Turing machine." However, only the physical ones will refute it. Unbounded idealizations, including analog computers with unbounded precision and accuracy, or those performing unbounded in time computations, do not count as empirical evidence, see Casares (E), and therefore they cannot refute Church's thesis.

¶4 · Under Church's thesis, complete languages are the most capable languages, that is, the most expressive ones, see Casares (H). Therefore, *under Church's thesis, undecidable propositions cannot be avoided.* Although this is very general, it still depends on Church's thesis being true. In order to determine the scope of validity of Church's thesis, we need to ascertain its nature. And here we follow Post (1936) program. We defend that Church's thesis is a consequence of the law of Post. The law of Post is a law of nature, and as such it can be refuted empirically, but, as long as it is not refuted, it provides general formulations of the incompleteness and unsolvability theorems that state some absolute human limitations:

*we humans cannot avoid undecidable propositions,*
*we humans cannot solve every problem,*
*human knowledge cannot be complete.*

## §9 Kant

¶1 · In my opinion, see Casares (K), the main point made by Kant is that all each subject knows is calculated by her brain. Except the raw data acquired by my senses, for example a red photon being detected at some point of my left retina, everything else is the result of some calculations done by my brain. Since we do not experience photons hitting our retinas, what we do experience, for example seeing a red stone, is the result of some calculations done by our brains. Therefore, what we take for real, as the red stone, is the result of some complex calculations done by our brains, implying that even the evidence that 'I am seeing a red stone now' is calculated by my brain.

¶2 · However, Kant seemed to like both to use a cryptic language and to make transcendental deductions, so I am possibly misunderstanding him. Nevertheless, this is immaterial as long as you accept that what is written in the previous paragraph is sensible save, perhaps, its attribution to Kant. In fact, we can ignore his deductions, because the law of Post is what Kant was looking for, since it states precisely the limits of the calculating capability of the human brain, but again I could well be wrong on Kant's intentions. And again this does not matter, provided you agree to name Post-Kantian subjectivism the position stating both *Kant's thesis*, which I interpret to say that

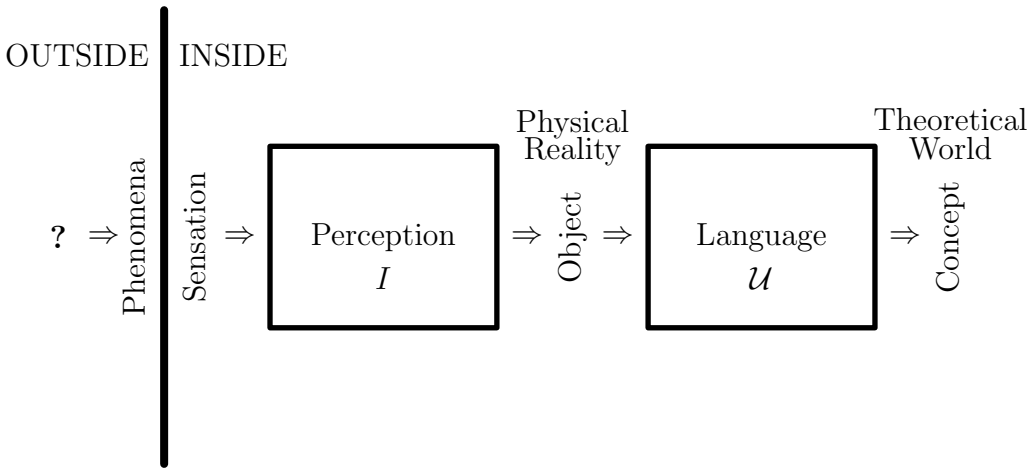
*all we subjects know is calculated by our brain*,

and the law of Post, now formulated to say that

*our human brain calculating capability is limited to universal computing*.

¶3 · Without Kant's thesis, the law of Post is the fundamental law of cognition. With Kant's thesis, in addition, the law of Post culminates an epistemological endeavor. Under Kant's thesis, the law of Post sets the limits of knowledge, and even the limits of all possible knowledge. The law of Post fits nicely with Kant!

¶4 · For us humans, physical reality and complete language are given *a priori*, in Kantian parlance. Although neither is operative when born, because both need some development, both are available some time later. After that point in time, every human investigation starts with a firmly-established physical reality, which we can observe and measure at will, and a full-self-expressible complete language, which is a finite calculating tool able to be used infinitely. It is in this way that both are given *a priori*. In other words, when we investigate anything, these are the two tools we have at our disposal.



¶5 · We cannot experience physical reality without our brain's perception, a name we use to refer to some brain modules. Therefore, perception is a condition of all possible experience. As my evidence is what I am certain about, then, following Descartes, my evidence is only what I experience in first person. Now, perception is a condition of all possible evidence.
¶6 · And *language is a condition of all possible theory*, since we cannot set any theory, including physics, without language. So our Kantian priority of the complete language is an epistemological priority. We are not saying, for example, that language is chronologically prior to life, or to the universe. However, without language the mere idea that life and language are somehow situated in time would be impossible; inconceivable is a better word, since there would not be any conceptual world without language.

## §10 Discussion

### §10.1 Physics

¶1 · Gödel's incompleteness theorem is limited to formal systems. In addition, if the number of propositions is finite, then (in principle) it is possible to list the truth value of all of them. Consequently, Gödel's incompleteness theorem is limited to formal systems that are not finite. In particular, given that the proof by Gödel (1930) is finitary, his theorem is limited to formal systems that are infinite enumerable.
¶2 · From this conclusion is easy to dismiss the impact of Gödel's incompleteness theorem. In one direction, we could argue that infinities do not really exist, and finite systems can be complete. And, in the other direction, we could say that physical reality is not enumerable, but continuous, as the real numbers, see §2, and then reality would be out of Gödel's incompleteness theorem scope. So physics should not be affected, although it is a formal system that includes arithmetic and that it has to be either consistent or useless, but not both.
¶3 · Computing by Turing (1936) introduces a new point of view on these matters. Instead of focusing on reality, now we center on language, and human language is certainly infinite enumerable, since it can "make infinite use of finite means". This is Humboldt's famous definition quoted by Chomsky (1965, in page 8) to introduce the generative grammars, which are Turing machines, see Casares (H). The definition remarks that both the number of rules of syntax and of words are finite, but the number of words in a sentence is unbounded, potentially infinite. And the same can be said of those formal systems where both the number of rules of inference and of axioms are finite, but the number of steps in a proof is unbounded, potentially infinite.
¶4 · So, from the wider Turing computing point of view, physics is affected simply because physics is a theory that has to be expressed in a complete language, which happens because arithmetic has to be expressed in a complete language, which is a consequence of Gödel's incompleteness theorem. Then, *physics cannot be complete*. This contrasts with physical reality, since all we can observe or measure is finite, that is, all we perceive are finite means used finitely. As an aside, noticing that physical measurements are never irrational real numbers, but always rational numbers, should imply that physical reality is not continuous, but discrete.

### §10.2 Paradoxes

¶1 · From the wider Turing computing point of view, it is easier to see the nature of paradoxes. For example, let us name the set of the halting computations of a complete language $H$, where a computation is defined by a pair of program and input coded data, that is, by a pair $\langle \mathfrak{p}|\vec{\mathfrak{d}} \rangle$ in the complete language $\mathcal{L}$ implemented by the universal Turing machine $\mathcal{U}$. In a sense, set $H$ is well defined, because the condition for a computation to belong to it is definite. In other words, the problem defining the set is perfectly determined. However, as shown by the diagonal argument in §6, we know that this particular problem is unsolvable. That is, we know of some computations that belong to the set, and of some other computations that do not belong to the set, but the exact extent of set $H$ is essentially indeterminate.

¶2 · It happens the same, as a consequence of Gödel's (1930) incompleteness theorem, to the set of all TRUE arithmetic propositions $T$: the exact extent of set $T$ is essentially indeterminate. So set $T$ is complete by fiat, but we would need to invoke an oracle in order to instantiate it. Note that oracles, see Turing (1938), are definable but not implementable, remember §6¶8, meaning that, whenever an oracle is needed, we are beyond what we can calculate.

¶3 · We will use Post (1944) terminology to characterize these sets: a set is *recursive* if and only if it is possible to recursively enumerate both the set and its complement, where recursion is synonymous with computing, see §7. Therefore, the extents of the recursive sets are exactly determined, since we can always decide what is in and what is out of them, but, as a consequence of Gödel's undecidability, there are sets that are not recursive, as $H$ and $T$, and the extents of the non-recursive sets are essentially indeterminate.

¶4 · They are *essentially* indeterminate because, as this happens in complete languages, the indeterminacy cannot be cured by using more expressive languages, since the complete languages are, under Church's thesis, the most expressive ones; see Casares (H). Then we should conclude that the indeterminacy is caused by limitations of the complete languages. So it is by implementing the liar paradox in diagonal arguments, see §2¶5, that the incompleteness and unsolvability theorems show some limitations of the complete languages, including human language if the law of Post is right. And given that, as Gödel (1930) writes in note 14 (see §6¶1), every epistemological antinomy can be used to prove a case of undecidability, then for each paradox we obtain its corresponding essentially indeterminate non-recursive set. In this way, *paradoxes are linguistic illusions*.

¶5 · Although we can refer to the non-recursive sets the same way we refer to the well-behaved recursive ones, this is an illusion: under the law of Post, the non-recursive sets are beyond our calculating capability. This is both marginal and fundamental:

- It is marginal because our calculating capability is more than enough for our everyday tasks. In fact, the corresponding complete language has made our species the most powerful and dangerous one, see Casares (T), showing that our calculating capability is much more than enough to survive; perhaps much too much!
- It is fundamental because sets are not only the basis of mathematics, but sets are also the very foundation of the whole theoretical world, since every conceptual definition is a predicate determining what is in and what is out of it. Being fundamental, it can cause basic misunderstandings. For example, the real numbers $\mathbb{R}$ are considered real, but $\mathbb{R}$ is a non-recursive set, see §2; consequently, that 'reality is continuous' is a linguistic illusion.

## §11 Conclusion

¶1 · Using Cantor's (1891) diagonal argument, see §2, Gödel (1930) showed that there are more formalized arithmetical propositions than enumerable proofs, see §3; therefore some propositions are undecidable since they can neither be proved TRUE nor FALSE using finitary formal systems. Using Cantor's diagonal argument, Turing (1936) showed that there are more definable problems than enumerable computations, see §6; therefore some problems are unsolvable by finitary machinery.

¶2 · The word 'finitary' is a bit weird, since it is neither 'finite' nor 'infinite', but midway between the two. *Finitary* refers to the infinite use of finite means. This point us twice to human language. Firstly, because that is precisely Humboldt's definition, see §10.1, and secondly, because out of the two Kantian tools, see §9¶4, physical reality is already much more limited since, as seen above, all we perceive are finite means used finitely. In any case, as the means are finite, the finitary system products are always enumerable.

¶3 · The finitary concept is best grasped by the Turing machine, see §4 and §5: the finite-state machine represents the finite means, and the unbounded tape allows the infinite use. And in Turing computing the most expressive languages are the complete languages implemented by universal Turing machines: a complete language implements the whole semantics of computing, which is thus full-self-expressible. This means that the most expressive languages that finitary systems can implement are the complete languages.

¶4 · Therefore, the diagonal argument is used to show some limitations of human language, see §10.2, and of other finitary systems, which derive from it. In these systems, the means are finite, but we do not limit their use. This is, of course, an idealization, but nevertheless it would also be wrong to limit, for example, the number of words in a sentence, or the number of steps in a proof. Summarizing, the incompleteness and unsolvability theorems by Gödel, Church, and Turing find limitations in our finitary tool, which is our complete language.

¶5 · And, in any complete language, there are definable concepts that cannot be expressed. This generalizes Gödel's incompleteness theorem, see §6¶8, which represents, under the law of Post, an absolute human limitation, see §7 and §8, which Kant promotes to transcendental, see §9. So, epistemologically, Gödel's incompleteness theorem elevates from Turing's generalization,

*every Turing complete language is Gödel incomplete*,

to its Post-Kantian formulation,

*knowledge cannot be complete*.

## References


Abelson & Sussman (1985): Harold Abelson and Gerald Jay Sussman with Julie Sussman, *Structure and Interpretation of Computer Programs*; The MIT Press, Cambridge MA, 1985, ISBN: 978-0-262-01077-1.

Cantor (1891): Georg Cantor, „Ueber eine elementare Frage der Mannigfaltigkeitslehre“; in *Jahresbericht der Deutschen Mathematiker-Vereinigung*, vol. 1, pp. 75–78, 1890–1891, https://gdz.sub.uni-goettingen.de/id/PPN37721857X_0001.

Casares (C): Ramón Casares, “Proof of Church’s Thesis”; DOI: 10.6084/m9.figshare.4955501, DOI: 10.48550/arXiv.1209.5036.

Casares (E): Ramón Casares, “Errors in Infinite Computations”; DOI: 10.6084/m9.figshare.13686439.

Casares (H): Ramón Casares, “A Complete Hierarchy of Languages”; DOI: 10.6084/m9.figshare.6126917.

Casares (K): Ramón Casares, “Subjectivist Propaganda”; DOI: 10.6084/m9.figshare.13076906.

Casares (T): Ramón Casares, “On Turing Completeness, or Why We Are So Many”; DOI: 10.6084/m9.figshare.5631922.

Chomsky (1965): Noam Chomsky, *Aspects of the Theory of Syntax*; The MIT Press, Cambridge MA, 1965, ISBN: 978-0-262-53007-1.

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.

Davis (1958): Martin Davis, *Computability & Unsolvability*; Dover Publications, New York, 1982, ISBN: 978-0-486-61471-7. Enlarged version of the work originally published by McGraw-Hill Book Company, New York, in 1958.

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.

Davis (1982): Martin Davis, “Why Gödel Didn’t Have Church’s Thesis”; in *Information and Control*, vol. 54, pp. 3–24, 1982, DOI: 10.1016/s0019-9958(82)91226-8.

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).

Kleene (1935a): Stephen Kleene, “General Recursive Functions of Natural Numbers”; in *Mathematische Annalen*, vol. 112, no. 1, pp. 727–742, December 1936, DOI: 10.1007/BF01565439. Presented to the American Mathematical Society, September 1935.

Kleene (1935b): Stephen Kleene, “$\lambda$-Definability and Recursiveness”; in *Duke Mathematical Journal*, vol. 2, pp. 340–353, 1936, DOI: 10.1215/s0012-7094-36-00227-2. Received July 1, 1935; presented to the American Mathematical Society, September 13, 1935.

Kleene (1952): Stephen Kleene, *Introduction to Meta-Mathematics*; Ishi Press, New York, 2009, ISBN: 978-0-923891-57-2. Reprint of the same title by North-Holland, Amsterdam, 1952.

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. Manuscript received May 6, 1955.

Moore (1956): Edward F. Moore, "Gedanken-Experiments on Sequential Machines"; DOI: 10.1515/9781400882618-006. In *Automata Studies* (editors C. E. Shannon and J. McCarthy), Volume 34 in the series Annals of Mathematics Studies (AM34); Princeton University Press, Princeton, 1956, pp. 129–153; ISBN: 0-691-07916-1.

Petzold (2008): Charles Petzold, *The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*; Wiley Publishing, Indianapolis, 2008, ISBN: 978-0-470-22905-7.

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

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.

Shagrir & Pitowsky (2003): Oron Shagrir and Itamar Pitowsky, "Physical Hypercomputation and the Church-Turing Thesis"; in *Minds and Machines*, vol. 13, pp. 87–101, 2003, DOI: 10.1023/A:1021365222692.

Turing (1936): A. M. 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): A. M. Turing, "Computability and λ-Definability"; in *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, December 1937, DOI: 10.2307/2268280.

Turing (1938): A. M. Turing, "Systems of Logic Based on Ordinals"; in *Proceedings of the London Mathematical Society*, vol. s2-45, no. 1, pp. 161–228, 1939, DOI: 10.1112/plms/s2-45.1.161. Received 31 May, 1938. Read 16 June, 1938.