A Complete Hierarchy of Languages

Ramón Casares

ORCID: 0000-0003-4973-3128

A complete hierarchy of languages is introduced to defend the thesis that 'human language is a Turing complete language'. This complete hierarchy discriminates languages on computability, meaning mixability, generativity, decidability, and expressiveness. We show that our thesis is true and, using the complete hierarchy, that our thesis is more specific and then more significant than other truths about human language. The complete hierarchy explains the evolution of language on the assumption that it was driven by expressiveness, and consequently it explains that our language is complete because complete languages are the most expressive languages.

Keywords: *Turing completeness, complete language, language hierarchy, language evolution, expressiveness.*

§1 Introduction

¶1 · In this paper we will defend a simple thesis:

Human language is a Turing complete language.

We start (in §2) by defining what is a (Turing) complete language (in §2.1), and by showing that the thesis is true (in §2.2).

This is DOI: 10.6084/m9.figshare.6126917.v2, version 20180825.

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

¶2 · Next, in order to compare the thesis on completeness with other truths about human language (which are listed in §3.1), we present a complete hierarchy of languages (in §3). The complete hierarchy starts from zero (in §3.2), and then goes to those languages with unmixable meanings, resulting in asyntactic languages with words but without sentences (in §3.3). Releasing the restriction on mixability, we get the syntactic languages, where the sentence meaning is calculated from the meanings of its words. Syntactic languages can be finite (in §3.4) or unbounded. Unbounded languages need a generative procedure, so they are also known as generative languages. When the number of words is unbounded, but the ways of mixing words is finite, we have the finitary languages (in §3.5). When both the number of words and the number of ways of mixing them are unbounded, but we require that every sentence is meaningful, we get the decidable languages (in §3.6). And releasing every requirement, we have the computable languages (in §3.7). Till this point, every new kind of language has properly included the previous one, and correspondingly every new kind of language was strictly more expressive than the previous one. However, as every language is computable, statements about computable languages are nonspecific and consequently uninteresting. Fortunately, there is a very interesting subset of the computable languages: the complete languages. The complete languages (in §3.8) are some syntactic, generative, and non-decidable languages that are strictly more expressive than any non-complete language. In fact, anything expressible in any language can be expressed in a complete language. And once the complete hierarchy is completed, we finish the section by drawing its Venn diagram (in §3.9).
¶3 · After having defined the complete hierarchy, it will be time to explore it (in §4). Firstly, we compare the complete hierarchy with the classical one by Chomsky (in §4.1). Next we show that the complete hierarchy can be used as a framework for the evolution of language, just by assuming that an individual who is able to express more meanings has more survival opportunities (in §4.2). Then, while we were adapting the concept of recursion to the complete hierarchy (in §4.3), we find another concept, 'hierarchical language', which cannot be adapted because it is independent of expressiveness. And lastly (in §4.4), we explain why the best definition of human language is our thesis, 'human language is complete': because it also says implicitly that human language is generative, unbounded, recursive, syntactic, and computable.
¶4 · Finally, we conclude with a summary (in §5).

§2 Turing completeness

§2.1 Definition

¶1 · A *Turing complete* device is a device that can be programmed to execute any computable function, save for time or tape limitations, where 'tape' is an abbreviation for 'external memory'. Not every computing device is Turing complete. For instance, while a full-programmable computer is Turing complete, an arithmetic calculator, which is one that can only compute the four arithmetic operations, is not Turing complete.
¶2 · Each Turing (1936) machine implements a computable function, where the Turing machine is the canonical model for computing devices, resulting that each piece of hardware implements a computable function. And the universal Turing machine is the canonical model for Turing complete devices, resulting that Turing completeness is the capacity of some hardware to calculate by software whatever hardware can calculate.

¶3 · The language used to program a Turing complete device is a Turing complete language, or a *complete language* for short. In other words, a language is *complete* if and only if any computable function can be expressed and calculated in it. In this second definition we have abstracted away the computer, but we should not forget that there is no language without a computing device.
¶4 · You can find more details on Turing completeness in Casares (2018).

§2.2 Truth

¶1 · The thesis *'Human language is a complete language'* is true because we can express and calculate any computable function in English, save for time or tape limitations and slips, where a 'slip' is a human error of execution. We are sure because Turing (1936) explains how to express and calculate any computable function in English.
¶2 · English is not the only human language, but we can extend the conclusion to other natural languages. Any language in which it is possible to explain how Turing machines work precisely is a complete language. How precisely? To pass this test, any person who speaks that language, after hearing the explanation, has to be able to imitate the calculation of any Turing machine on any data exactly, save for time or tape limitations and slips. In most natural languages, if not in all, it is possible to explain how Turing machines work to this level of detail, and therefore most natural languages, if not all, are complete.
¶3 · In addition, assuming that any human baby reared in an English speaking community will acquire English, being English a complete language, is assuming that the human brain is a Turing complete device, because otherwise it could not implement a complete language. This means that the native language of the human brain is a complete language, even if some natural languages were not complete.

§3 A hierarchy of languages

§3.1 Comparison

¶1 · A truth can be trivial, uninteresting, irrelevant or just a consequence of something more significant. Regarding language, we are defending here that completeness is the characteristic feature of human language, instead of syntax, or instead of recursion, which may seem more obvious characteristics. Then, our task now will be to compare the respective worth of these truths on human language: human language is syntactic, human language is generative, human language is unbounded, human language is hierarchical, human language is computable, human language is recursive, and human language is complete. In order to do this comparison, we will firstly present a series of languages.

§3.2 Null

¶1 · In the bottom of the hierarchy of languages we find no language. We will call any language that has only one unmixable meaning a *null language* $\mathcal{L}_0$. Having only one meaning, there is no need to select a meaning, and then there is no information to transmit, according to the theory of communication by Shannon (1948).
¶2 · Every language requires a computing device, and, in the case of a null language, the computer is a mechanism that always behaves the same way, so we will say that it has only one behaviour. It is then something that we will not usually consider a computing device, as a light bulb, for example, or a washing machine with a single program.

§3.3 Asyntactic

¶1 · An *asyntactic language* $\mathcal{L}_a$ is a language with a finite number of unmixable meanings. In this case, there is a bijection between the finite set of meanings and the finite set of words.

¶2 · The typical computing device implementing an asyntactic language $\mathcal{L}_a$ is a machine that can behave in a finite number of ways, when each way is selected by pressing a different button. An example is a washing machine with three washing programs that are selected by pressing one out of three buttons: one for the light program, another for the medium program, and the third for the strong program. Here, the button is the word, and the corresponding machine behaviour is its meaning.

¶3 · Every null language is an asyntactic language, because number one is a finite number, so the set of all asyntactic languages $\{\mathcal{L}_a\}$ includes the set of all null languages $\{\mathcal{L}_0\}$. In fact, the set of null languages is a proper subset of the asyntactic languages, $\{\mathcal{L}_0\} \subset \{\mathcal{L}_a\}$, because there are asyntactic languages with more than one meaning, as shown by the three programs washing machine language.

¶4 · It is more important that asyntactic languages are strictly more expressive than null languages, noted $\{\mathcal{L}_a\} \succ \{\mathcal{L}_0\}$, or $\{\mathcal{L}_0\} \prec \{\mathcal{L}_a\}$. This is because, given any null language with its only meaning m_0, we can always construct an asyntactic language that is more expressive: that with, at least, meanings m_0 and m_1, for any meaning m_1 such that $m_1 \neq m_0$. It can *always* be *more* expressive, hence the 'strictly'.

§3.4 Finite

¶1 · A *finite language* $\mathcal{L}_f$ is a language with a finite number of meanings. As now some meanings can be mixed, we can get a new meaning by mixing some mixable meanings. Suppose, for example, that we take two mixable meanings, let us call them m_1 and m_2, each one with its corresponding word, let us call them w_1 and w_2. Then the easiest way to represent the meaning that results from mixing m_1 and m_2 will be the sentence $w_1 w_2$. With words and sentences, this language is syntactic.

¶2 · To keep the number of meanings finite, the number of unmixable meanings and the number of mixable meanings have to be finite, and the ways of mixing meanings, also known as operations, have to be decidable and finite in number.

¶3 · A simple case of finite language that is not asyntactic is that implemented by a washing machine where you can select independently: one out of four program times, one out of three temperatures of the water, and whether to apply a fast or a slow final spin cycle. This language has $4 + 3 + 2 = 9$ words (buttons), $4 \times 3 \times 2 = 24$ well-formed word combinations, also known as sentences, and then 24 meanings (behaviours).

¶4 · Words are atomic (= indivisible) mixable meanings, but only sentences have full meaning. Following this convention, if a separate word has full meaning, then that single word by itself makes a sentence. Then, any unmixable meaning results in a word that makes a sentence by itself, as some interjections do. Usually, in syntactic languages, most words are mixable meanings, though they can include some archaic unmixable meanings.

¶5 · Every asyntactic language is finite, but not the converse, $\{\mathcal{L}_a\} \subset \{\mathcal{L}_f\}$, because $\mathcal{L}_f$ meanings are not restricted to be unmixable, as $\mathcal{L}_a$ meanings are. Asyntactic languages equate a meaning to a word, so they do not need sentences, while finite syntactic languages equate a meaning to a sentence because there are mixable meanings in these languages. So, from the syntactic point of view, every single word of the asyntactic language makes a sentence.

¶6 · Finite languages are strictly more expressive than asyntactic languages, $\{\mathcal{L}_f\} \succ \{\mathcal{L}_a\}$. Proof: given any asyntactic language, we can construct a finite language which is more expressive; just let that each meaning of the asyntactic language results in a one-word sentence in the finite one, and then add a pair of mixable meanings to the finite language. Quantitatively, going from an asyntactic to a finite syntactic language, the number of meanings typically increases exponentially, because the number of meanings in a syntactic language results from multiplying the number of mixable meanings, so, for example, given w words, we can make w^2 pairs, w^3 triplets, and, in general, w^n n-tuples.

§3.5 Finitary

¶1 · Releasing the requirement of finiteness we get the unbounded languages. To go beyond finiteness, a generative procedure is required, so unbounded languages are also known as generative languages. As in practice the infinite is out of reach, whenever a generative process is involved we should be aware that it works save for time or tape limitations.

¶2 · The simplest case is when the set of words is infinite, but the number of ways of mixing them is finite. A *finitary language* $\mathcal{L}_\aleph$ is any language that has an unbounded set of words, and a finite set of operations, where an operation is a decidable way of mixing words, and a word is an atomic meaning.

¶3 · With just one mixable meaning, the number one represented by the symbol '|', and one mixing operation, string concatenation, we can already express every natural number in bijective base one.

$$S \to \mathsf{I} \mid S\,\mathsf{I}$$

In this language, the string '|||||' means the number five. Though being very simple, this finitary language is not finite, and then it already needs a generative procedure, and astronomically big numbers will meet time and tape limitations.

¶4 · A more interesting finitary language that is not finite is the language implemented by a basic arithmetic calculator.

$$\begin{aligned} S &\to N\,O\,N\,= \\ N &\to D \mid N\,D \\ D &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ O &\to + \mid - \mid \times \mid \div \end{aligned}$$

Using any standard definition of the arithmetic operations O on the decimal natural numerals N, we can express and calculate any addition, subtraction, multiplication or division of two numbers in this language. For example, $3 + 4$ and 2×7.

¶5 · Every finite language is finitary, but some finitary languages are not finite, that is, $\{\mathcal{L}_f\} \subset \{\mathcal{L}_\aleph\}$. This is because a finitary language is a finite language but with one requirement released.

¶6 · Finitary languages are strictly more expressive than finite languages, $\{\mathcal{L}_\aleph\} \succ \{\mathcal{L}_f\}$. Proof: given any finite language, we can always build a finitary language that includes the finite one completely and, in addition, all the natural numbers. Quantitatively, going from a finite to a generative language, the number of meanings typically increases infinitely, because it goes from a finite to an infinite enumerable number ($\aleph_0$).

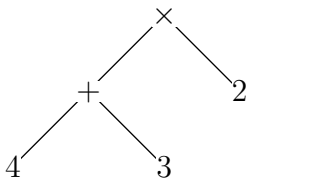
§3.6 Decidable

¶1 · A *decidable language* $\mathcal{L}_D$ is any language in which every sentence has a meaning. In other words, there are not paradoxes in decidable languages, where a *paradox* is a sentence that never reaches a meaning, because its calculation gets stuck in an infinite loop. While in finitary languages the number of operations for mixing meanings is finite, in decidable languages this condition is released.

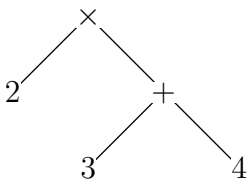
¶2 · The language implemented by a reverse Polish notation arithmetic calculator, or RPNA language for short, is a decidable language that is not finitary.

$$
\begin{aligned}
&S \to N, \mid SN, \mid S\,S\,O \\
&N \to D \mid ND \\
&D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&O \to + \mid - \mid \times \mid \div
\end{aligned}
$$

Using any standard definition of the arithmetic operations O on the decimal natural numerals N, we can express any unconditional arithmetic calculation in this language.



$$4, 3, +\,2, \times = 14$$
$$(4 + 3) \times 2 = 14$$



$$2, 3, 4, +\times = 14$$
$$2 \times (3 + 4) = 14$$

¶3 · The precise demarcation of decidable languages, which is the boundary separating decidable from non-decidable languages, is not itself decidable, as a consequence of the halting problem, see Davis (1958), page 70. In spite of this, we can show that some languages are decidable. For example, any finitary language is decidable, because the restriction to a finite set of mixing operations precludes any loop, and then any infinite loop. Primitive recursion is also decidable, because all of its loops do decrease a variable and there is a bottom, which is 0, so they are never infinite.

¶4 · Every finitary language is decidable, as just seen above, but some decidable languages are not finitary, as shown by the RPNA language, and then $\{\mathcal{L}_\aleph\} \subset \{\mathcal{L}_D\}$.

¶5 · Decidable languages are strictly more expressive than finitary languages, that is, $\{\mathcal{L}_D\} \succ \{\mathcal{L}_\aleph\}$. For example, in the finitary language of the basic calculator we can compute $3 + 4$ and 2×7, but we cannot compute $2 \times (3 + 4)$, as we can do in the decidable language of the RPNA calculator. Proof: given any finitary language, we can build a decidable one that includes the finitary language completely, and then add a generative and decidable process to compose operations.

§3.7 Computable

¶1 · Releasing all the requirements, that is, those on the mixability of meanings, those on the finitude of words, those on the finitude of operations, and those on the existence of meaningless sentences, we get the computable languages. A *computable language* $\mathcal{L}_T$ is any language that any computing device can implement, and particularly any Turing machine. Then a computable language can also be called a Turing language $\mathcal{L}_T$.

¶2 · Every decidable language is computable, but some computable languages are not decidable, $\{\mathcal{L}_D\} \subset \{\mathcal{L}_T\}$. This is because every language, and then every decidable language, is computable, but there are computable languages with paradoxes, that is, with infinite loops, and these languages are not decidable. That 'every language is computable' is a consequence of Church's (1935) thesis, which we assume.

¶3 · A paradox is a sentence that never reaches a meaning, because its calculation gets stuck in an infinite loop. Then, as paradoxes are meaningless, it could seem that there is some decidable language that contains every meaning. This is not the case as a consequence of the halting problem. Therefore, decidable languages are strictly less expressive than computable languages, $\{\mathcal{L}_D\} \prec \{\mathcal{L}_T\}$. Note that this result is a reformulation of Gödel's (1930) incompleteness theorem: there is not any decidable language that contains every mathematically meaningful truth. Or, in Post (1944) form: there is a recursively enumerable set that is not recursive.

§3.8 Complete

¶1 · Please pay attention now, because the next step is of a different kind. Till now, we were releasing requirements step by step and, as a result, we were getting more expressive languages on each step. And, after releasing every requirement, we have reached the computable languages. However, as every language is computable, statements about computable languages are nonspecific and then uninteresting. In particular, computable languages include the most expressive languages, but also those that do not need expressions, as the null languages, and also every language in between. For example, in the RPNA computable language, we can calculate $(((1 \times 2) \times 3) \times 4) \times 5$, but we cannot express:

$$n, r := 2, 1;\ \text{while } n < 6\ \{r :=\times n;\ n{+}{+}\};\ \text{return } r.$$

This translates to English as: firstly assign values 2 and 1 to variables n and r, respectively; then and while the value of variable n is less than 6, keep multiplying the value of variable r by the value of variable n, and increasing the value of variable n by one; and finally return the value of variable r.

¶2 · So now, instead of releasing a requirement, we will impose a new one. A *complete language* $\mathcal{L}_C$ is any language that has the maximum expressiveness. This definition is equivalent to that on §2.1, because being able to execute any computable function is being able to compute anything that any computer can compute, and then, as any language needs a computer, anything that any language can express and calculate can be expressed and calculated in the complete language.

¶3 · As shown in the previous example, conditional loops are beyond the capabilities of the RPNA computable language. It is also clear in the example that to take advantage of conditional loops it is important to handle words with a variable meaning, such as pronouns. And to handle these words that do not have a fixed meaning, the computing device has to treat them as if they had any meaning, or even no meaning, that is, they have to be computed independently of their meanings.

¶4 · The ability of using words without meaning could seem unimportant, but those words allow us to ask questions. Suppose, for example, that I say to you: 'who is reading the book?' Though the sentence is syntactically complete, it uses the interrogative pronoun 'who', which is a word without a fixed meaning, and then its semantics is not complete, so I am asking you to complete its meaning. In that case you can answer just 'Inés', which is perhaps not as syntactically complete as 'Inés is reading the book', but anyway your short answer completes the semantics exactly as if the full answer were said.

¶5 · To work with words independently of their meanings, complete languages need a peculiar type of words: functional words, as 'while' in the previous example. The meaning of these functional words is just syntactical, as they work on syntactic structures, or on words independently of their meanings, conforming to some functional semantics. And not any functional semantics goes.

¶6 · By the equivalent definition in §2.1, a *complete language* $\mathcal{L}_C$ is any language in which any computable function can be expressed and calculated. Therefore, for any language to be complete, it has to fulfil two requirements.

- Syntactic requirement for completion: the number of its syntactic structures has to be infinite enumerable. Turing (1936) showed that the number of Turing machines, and then the number of computable functions, is infinite enumerable, so this requirement grants us that we can assign a different syntactic structure to each and every computable function.
- Semantic requirement for completion: the Turing complete device, taking any such syntactic structure expressing a computable function, has to calculate that function. Therefore, its functional semantics has to be designed precisely to make this happen.

¶7 · As some computable functions are partial, it is possible to express infinite loops in every complete language, and then in any complete language there are paradoxes. And conversely, if it is not possible to express any paradox in a language, then that language is not complete. Therefore, no complete language is decidable, $\{\mathcal{L}_C\} \cap \{\mathcal{L}_D\} = \emptyset$.

¶8 · As a consequence of Gödel (1930) numberings, full reference is a property of every complete language. And using self reference, included in full reference, it is easy to concoct a paradox: 'this sentence is false'. Theorem :-) In every language that is expressive enough to mean 'this sentence is false', there is a paradox. Then, the sentence "'this sentence is false' is a paradox" is true in English, but it cannot be expressed in any decidable language. This shows that English is not decidable.

¶9 · Some (or rather most) computable languages are not complete. For example, the RPNA language implemented by the reverse Polish notation arithmetic calculator is computable, but it is not complete. The RPNA language is not complete because all of its well-formed expressions are decidable, that is, because we can not express any paradox in it as it is not possible to express infinite loops in it. However, the converse is true, that is, every complete language is computable. This is because every language is computable, as no requirement is imposed on computable languages. Therefore, $\{\mathcal{L}_C\} \subset \{\mathcal{L}_T\}$.

¶10 · By definition, complete languages are the most expressive languages, meaning that everything that can be expressed in any language can also be expressed in a complete language. Then, complete languages are more expressive than, or equally expressive to, computable languages, $\{\mathcal{L}_C\} \succeq \{\mathcal{L}_T\}$. They are not always more expressive because some computable languages are complete, $\{\mathcal{L}_T\} \cap \{\mathcal{L}_C\} = \{\mathcal{L}_C\} \neq \emptyset$.

§3.9 Complete hierarchy

¶1 · So there is a progression in expressiveness

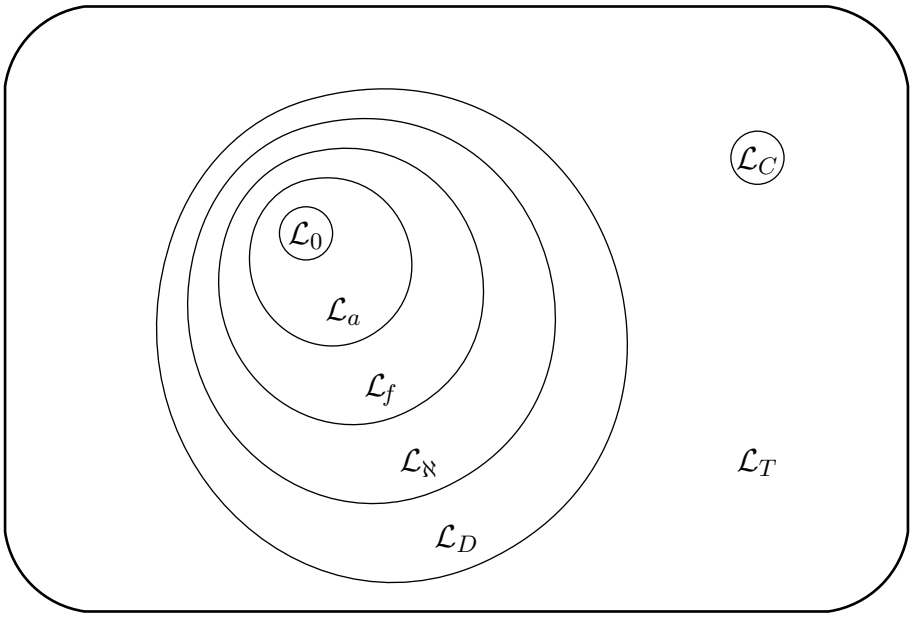
$$\{\mathcal{L}_0\} \prec \{\mathcal{L}_a\} \prec \{\mathcal{L}_f\} \prec \{\mathcal{L}_\aleph\} \prec \{\mathcal{L}_D\} \prec \{\mathcal{L}_T\} \preceq \{\mathcal{L}_C\}$$

which is not replicated in inclusiveness

$$\{\mathcal{L}_0\} \subset \{\mathcal{L}_a\} \subset \{\mathcal{L}_f\} \subset \{\mathcal{L}_\aleph\} \subset \{\mathcal{L}_D\} \subset \{\mathcal{L}_T\} \supset \{\mathcal{L}_C\}$$

because there is an inversion in the last step, from computability to completeness.

¶2 · Because of the inversion, the hierarchy of languages is defined by expressiveness, and not by inclusiveness, which instead, together with $\{\mathcal{L}_C\} \cap \{\mathcal{L}_D\} = \emptyset$, draws the following Venn diagram.



Language
Null $\mathcal{L}_0$
Asyntactic $\mathcal{L}_a$
Finite $\mathcal{L}_f$
Finitary $\mathcal{L}_\aleph$
Decidable $\mathcal{L}_D$
Computable $\mathcal{L}_T$
Complete $\mathcal{L}_C$

¶3 · This hierarchy is complete because it spans from the null languages $\mathcal{L}_0$, where no expression is needed, to the complete languages $\mathcal{L}_C$, where everything can be expressed.

¶4 · We will extend our notation using set difference, where $A \setminus B$ is the set of the elements in set A but not in set B:

$$\{\mathcal{L}_{A\setminus B}\} = \{\mathcal{L}_A\} \setminus \{\mathcal{L}_B\}.$$

For example, using this convention, $\mathcal{L}_{f\setminus a}$ is a finite language that is not asyntactic, so it is a finite syntactic language. And now we can express that $\{\mathcal{L}_C\} \succ \{\mathcal{L}_{T\setminus C}\}$.

¶5 · We can read this hierarchy in several ways. From the syntactical point of view, we have three main classes of languages:

- The null languages $\mathcal{L}_0$, which neither need words nor sentences.
- The pure asyntactic languages $\mathcal{L}_{a\setminus 0}$, which do not need sentences, and for which a meaning is equivalent to a word.
- All other languages $\mathcal{L}_{T\setminus a}$ are syntactic. A *syntactic language* $\mathcal{L}_{T\setminus a}$ is any that is not asyntactic. Syntactic languages need sentences and words, where a word is an atomic meaning, a sentence is a mix of words, and a meaning is equivalent to a sentence.

¶6 · We can also read the hierarchy from the generative point of view, and then we find two main classes of languages. The finite languages are $\mathcal{L}_f$. And the unbounded languages are the rest, $\mathcal{L}_{T\setminus f}$. The unbounded languages need a generative process, so a *generative language* $\mathcal{L}_{T\setminus f}$ is any that is not finite.

§4 Discussion

§4.1 Chomsky hierarchy

¶1 · It seems appropriate to compare the hierarchy presented here with the classical one by Chomsky (1959). We will refer to his paper throughout this subsection.

¶2 · Chomsky starts without any restriction, so his type 0 language $\mathcal{T}_0$ is any generated by any Turing machine, see page 143, that is, a type 0 language $\mathcal{T}_0$ is any computable language $\mathcal{L}_T$. Therefore, $\{\mathcal{T}_0\} = \{\mathcal{L}_T\}$.

¶3 · Chomsky starts without any restriction and in each step he adds a new restriction, and then his hierarchy goes in the opposite direction to the one presented in this paper.

$$\{\mathcal{T}_0\} \supset \{\mathcal{T}_1\} \supset \{\mathcal{T}_2\} \supset \{\mathcal{T}_3\}$$

Direction is not important. What is relevant is that his hierarchy is based on inclusiveness, instead of being based on expressiveness, as it is our complete hierarchy.

¶4 · In his theorem 3, page 143, Chomsky states that each type 1 language $\mathcal{T}_1$ is decidable, but not conversely, see note 7a, and then $\{\mathcal{T}_1\} \subset \{\mathcal{L}_D\}$.

¶5 · The last type of languages in Chomsky hierarchy is type 3, $\mathcal{T}_3$, which are the regular, or finite state languages, see his theorem 6, page 150. As these languages are generative $\mathcal{L}_{T\setminus f}$, we have that $\{\mathcal{L}_f\} \subset \{\mathcal{T}_3\}$.

¶6 · In summary:

- Chomsky hierarchy does not include our languages $\mathcal{L}_0$, $\mathcal{L}_a$, and $\mathcal{L}_f$, because they are not generative.
- Chomsky hierarchy does not include our complete language $\mathcal{L}_C$, because his hierarchy is inclusive and blind to language expressiveness.
- His type 0 is our computable language, $\mathcal{T}_0 = \mathcal{L}_T$, and his other types $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, correspond to our finitary language $\mathcal{L}_\aleph$.
- Our decidable language $\mathcal{L}_D$ is between his types $\mathcal{T}_0$ and $\mathcal{T}_1$.

¶7 · Therefore, we can combine Chomsky hierarchy and the complete hierarchy, as follows.

$$\{\mathcal{L}_0\} \subset \{\mathcal{L}_a\} \subset \{\mathcal{L}_f\} \subset \{\mathcal{T}_3\} \subset \{\mathcal{T}_2\} \subset \{\mathcal{T}_1\} \subset \{\mathcal{L}_D\} \subset \{\mathcal{T}_0\} = \{\mathcal{L}_T\} \supset \{\mathcal{L}_C\}$$

§4.2 The complete hierarchy and evolution

¶1 · If human language is generative, which it is, and its syntax is decidable, which it is, then Chomsky hierarchy is perfectly focused to study human language syntax as it is now. For example, some interesting results by Stabler (2014) locate human grammar between context free $\mathcal{T}_2$ and context sensitive $\mathcal{T}_1$ grammars. On the other hand, generative syntax is not the whole of language and in our complete hierarchy we can go further.

¶2 · In one direction because, by including some non-generative languages, the complete hierarchy can be used to locate the first steps in the evolution of human language. In particular, protolanguage as proposed by Bickerton (1990) is an asyntactic language $\mathcal{L}_a$. This is a convincing first step, because other animals communication systems are also asyntactic languages $\mathcal{L}_a$. And a finite syntactic language $\mathcal{L}_{f\setminus a}$, as the language proposed by Progovac (2016), would be a convincing exponentially more expressive second step.

¶3 · In the other direction because, by dealing with meanings, the complete hierarchy deals with semantics, so it can discriminate paradoxes from meaningful sentences, and most importantly, it can compare the expressiveness of languages, and thus it can identify

and denote the most expressive languages, which are the complete languages. In this case, a complete language $\mathcal{L}_C$ would be the last step in the evolution of human language.
¶4 · This evolutionary argument is valid under the assumption that there is a direct relationship between expressiveness and evolutionary fitness, at least in some environments. This would mean that, in those environments, any individual who is able to express more meanings would have more survival opportunities, which is a sensible possibility.
¶5 · If you find that meaning is a too imprecise concept to be of any value, you can restrict meanings to solutions. This restriction is feasible because every solution is meaningful for its problem, and it could be wide enough considering that, in the end, everything that we living beings do, we do it to keep us living, which is to say that everything we do is to resolve the survival problem and its subproblems. If you accept the restriction, then you have to accept that the survival problem is the final source of every meaning. However, you can believe on the direct relationship between expressiveness and evolutionary fitness on other grounds, and that is all that is needed here.

§4.3 Recursion

¶1 · There is a concept considered very close to language which was only mentioned briefly when developing the complete hierarchy, and which is also alien to Chomsky hierarchy: recursion. What is a recursive language?
¶2 · In mathematics, computing is equivalent to recursion, because Turing (1937) proved that *every recursive function is computable and every computable function is recursive.* As each Turing (1936) machine implements a computable function, and the Turing machine is the mathematical model of a computing device, then each piece of hardware implements a computable function. Therefore, the theorem says that, save for time or tape limitations, the set of all functions that can be implemented by hardware is equal to the set of all recursive functions.
¶3 · In computing, or at least for those following Post (1944), a recursively enumerable set is a computable set, and a recursive set is a decidable set. With this finer distinction, he could state that: "There exists a recursively enumerable set of positive integers which is not recursive." Nevertheless, under Post definition of recursion as decidability, it is not true that human language is recursive, and therefore we will ignore his definition here.
¶4 · In linguistics, at least as defined by Watumull et al. (2014), a recursive function has to be computable, hierarchical, and unbounded. In their definition, Watumull et al. (2014) have introduced a new concept: hierarchical language. Of course, when they say that human language is hierarchical, they do not mean that human language has to be in any hierarchy, but that its syntactic structures have to be hierarchical. These hierarchical structures, which are also known as trees, are in opposition to the linear structures, which are also known as strings.
¶5 · In this paper, we have not yet dealt with the requirement of working on trees because this requirement is orthogonal to the others. It happens that computer generated trees can always be converted into strings by computable means in such a way that the original trees can be recovered from those converted strings by computable means. In other words, it is always possible to convert trees to strings and back by computing. Depending on the problem, it can be more efficient to work on trees than to work on strings, or the other way around, but it is always possible to do it either way. In fact, lambda-calculus, which is one of the two main mathematical models of computing, uses trees, while the canonical one by Turing (1936) uses strings, and they are equivalent, as proved by Turing (1937).

¶6 · As a consequence, there are complete languages that work on strings, and there are complete languages that work on trees. The only syntactic requirement is that the number of sentences of a complete language has to be infinite enumerable, see §3.8. And using the definition of generative language in §3.9, there are also generative languages that work on strings and generative languages that work on trees.

¶7 · The example used to illustrate the decidable languages in this paper, which was the RPNA language presented in §3.6, works on trees. Or rather, using Chomsky (1959) hierarchy, the RPNA language is context-free $\mathcal{T}_2$, and not a regular language $\mathcal{T}_3$ that only works on strings. In any case, being computable, hierarchical, and unbounded, the RPNA language is recursive for Watumull et al. (2014).

¶8 · In summary: For Turing (1937) recursion is computing, and then a recursive language would be a computable language $\mathcal{L}_T$, while for Watumull et al. (2014) a recursive language would be a computable, unbounded (= not finite) language $\mathcal{L}_{T\setminus f}$ that is hierarchical. In order to proceed, in what follows we will consider that a recursive language is a generative language $\mathcal{L}_{T\setminus f}$. This seems a just resolution, because it is one requirement away from both parties involved: it detracts the hierarchical requirement from one party, and it adds the generative requirement to the other. If you would ask me, I do personally prefer to define a recursive language as a complete language $\mathcal{L}_C$, because it is only in a complete language that each and every recursive function can be expressed and calculated.

§4.4 Human language is complete

¶1 · Finally, we can determine what is the best way to define our language. In §3.1 we listed several truths about human language, which we repeat here, but now with their corresponding nomenclature.

- Human language is syntactic $\mathcal{L}_{T\setminus a}$.
- Human language is generative $\mathcal{L}_{T\setminus f}$.
- Human language is unbounded $\mathcal{L}_{T\setminus f}$.
- Human language is hierarchical.
- Human language is computable $\mathcal{L}_T$.
- Human language is recursive $\mathcal{L}_{T\setminus f}$.
- Human language is complete $\mathcal{L}_C$.

After our analysis, we find that three of them are synonyms: generative, unbounded, and recursive. And one, hierarchical, is out of the complete hierarchy. Then only four of the truths will compete for the best definition of human language.

¶2 · As all are true, if one is more specific than another, then the more specific one makes a better definition. For example, 'human language is computable' is less specific than 'human language is generative', because $\{\mathcal{L}_T\} \supset \{\mathcal{L}_{T\setminus f}\}$, so the last definition is better. In fact, every language is computable, so even other animals communication systems qualify as computable languages.

¶3 · Therefore, after locating the four competing kind of languages, which are $\mathcal{L}_{T\setminus a}$, $\mathcal{L}_{T\setminus f}$, $\mathcal{L}_T$, and $\mathcal{L}_C$, in the Venn diagram (§3.9), the resolution is simple.

$$\{\mathcal{L}_T\} \supset \{\mathcal{L}_{T\setminus a}\} \supset \{\mathcal{L}_{T\setminus f}\} \supset \{\mathcal{L}_C\}$$

Complete languages are generative, generative languages are syntactic, and syntactic languages are computable, so 'human language is complete' is the most specific definition,

and then the best definition, out of those on discussion. In other words, saying that 'human language is complete', we are also saying implicitly that it is generative, unbounded, recursive, syntactic, and computable.

¶4 · Being an orthogonal feature, all four kind of languages in our contest can be hierarchical or not, so we can be more specific: human language is complete and hierarchical. Being orthogonal means that the reason why our language is hierarchical is not the reason behind the complete hierarchy, which is expressiveness. If our evolutionary path was one of increasing expressiveness, then the complete hierarchy, in which languages are ordered by their expressiveness, models the evolution of our language, and thus it explains that our language is complete because complete languages are the most expressive ones. But that argument cannot explain why human language is hierarchical, because hierarchical languages are not more expressive than non-hierarchical languages.

¶5 · Although the complete hierarchy cannot explain why our language is hierarchical, it can explain why it is syntactic, generative, non-decidable, and complete. Each of these features would be the mark of an evolutionary achievement. And the complete hierarchy suggests that the evolution of human language could have gone through a series of phases of increasing expressiveness: firstly asyntactic $\mathcal{L}_a$, next finite syntactic $\mathcal{L}_{f\setminus a}$, later generative decidable $\mathcal{L}_{D\setminus f}$, and finally complete $\mathcal{L}_C$.

Language		**Comment**	**Expressiveness**
$\mathcal{L}_a$	Asyntactic	Animal communication system	w meanings
$\mathcal{L}_{f\setminus a}$	Finite	Syntactic or non-asyntactic	w^n meanings
$\mathcal{L}_{D\setminus f}$	Decidable	Generative or non-finite	$\aleph_0$ meanings
$\mathcal{L}_C$	Complete	Non-decidable, but $\mathcal{L}_{C\setminus D} = \mathcal{L}_C$	All meanings

$$\{\mathcal{L}_a\} \prec \{\mathcal{L}_{f\setminus a}\} \prec \{\mathcal{L}_{D\setminus f}\} \prec \{\mathcal{L}_C\}$$

§5 Conclusion

¶1 · We conclude that 'human language is complete' is a better definition for human language than other possible definitions, as human language is generative, or unbounded, or recursive, or syntactic, or computable, because these other definitions, while also true, are less specific. In order to compare these possible definitions, we have developed a complete hierarchy of languages, where these kind languages and some others were defined precisely, so they could be compared precisely.

¶2 · We can be more specific: 'human language is complete and hierarchical'. However, this additional specification about the hierarchical nature of human language is out of the complete hierarchy, because the hierarchical feature cannot be defined using the discriminating concepts of the complete hierarchy, which are computability, meaning mixability, generativity, decidability, and expressiveness. Why human language is hierarchical has to be explained using other reasons, probably non-linguistic reasons, it seems to me.

¶3 · Nevertheless, the complete hierarchy can explain why our language is syntactic, generative, non-decidable, and complete, since it provides a framework to locate some phases of the evolution of human language that the classical hierarchy by Chomsky (1959) does not provide. Explaining language evolution using the complete hierarchy only requires us to assume that evolutionary fitness has a direct relationship with expressiveness.

References

Bickerton (1990): Derek Bickerton, *Language & Species*; The University of Chicago Press, Chicago, 1990, ISBN: 978-0-226-04611-2.

Casares (2018): Ramón Casares, "On Turing Completeness, or Why We Are So Many"; DOI: 10.6084/m9.figshare.5631922.v3.

Chomsky (1959): Noam Chomsky, "On Certain Formal Properties of Grammars"; in *Information and Control*, vol. 2, no. 2, pp. 137–167, June 1959, DOI: 10.1016/S0019-9958(59)90362-6.

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, NY, 1982, ISBN: 978-0-486-61471-7. This is an enlarged version of the original book of the same title published by McGraw-Hill, NY, 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.

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).

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.

Progovac (2016): Ljiljana Progovac, "A Gradualist Scenario for Language Evolution: Precise Linguistic Reconstruction of Early Human (and Neandertal) Grammars"; in *Frontiers in Psychology*, vol. 7, art. 1714, 2016, DOI: 10.3389/fpsyg.2016.01714.

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.

Stabler (2014): Edward Stabler, "Recursion in Grammar and Performance", DOI: 10.1007/978-3-319-05086-7_8; in *Recursion: Complexity in Cognition*, Volume 43 of the series 'Studies in Theoretical Psycholinguistics', Tom Roeper, Margaret Speas (editors), pp. 159–177, Springer, Cham, Switzerland, 2014, ISBN: 978-3-319-05085-0.

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.