# Universal Grammar is a universal grammar

**Ramón Casares**

ORCID: 0000-0003-4973-3128

*Language is the most distinctive feature of humans, but there is no consensus on what is characteristic of language. From the point of view of computing, we argue that 'the human brain circuitry that implements language is Turing complete'. This thesis makes evolutionary sense, and natural languages are expressive enough, but two issues against it remain: natural language syntax is decidable, and not every possible language can be a natural language. We answer the first showing that the syntax of a complete language can be decidable, and blaming functional semantics for the undecidability, where functional semantics is the semantics of syntax. To answer the second we distinguish native language, first language, and later languages, where all natural languages are first languages acquired during a critical period that eases the process by preventing some possibilities. The thesis supports the weak version of the linguistic relativity hypothesis, and explains the role played by language in the cognitive gap that separates our species from the rest.*

*Keywords:* *Universal Grammar, Turing completeness, language evolution*

## §1 Introduction

¶1 · The word 'universal' is used differently in computing, as in 'universal Turing machine', and in linguistics, as in 'Universal Grammar'. But: *Could it be that both 'universals' are nevertheless the same?*

This is DOI: 10.6084/m9.figshare.4956764, version 20170501.

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

¶2 · From Chomsky (1959), we deduce that for each universal Turing machine there is a universal grammar, and conversely, meaning that a universal grammar is equivalent to a universal Turing machine, §2.1. Meanwhile in linguistics, following Chomsky (2005), Universal Grammar is the human brain circuitry that implements the faculty of language, §2.2. So the definitive resolution is achieved only when we show that the human brain has the computing capacity of a universal Turing machine, according to Turing (1936), §2.3, and that language uses this capacity completely, §2.4. Then, Universal Grammar is a universal grammar, and our answer is: *yes*.

¶3 · The thesis that Universal Grammar is a universal grammar can also be formulated saying that Universal Grammar is Turing complete. In any case, the thesis implies that Universal Grammar implements a complete language, which is undecidable. We find that the syntax of a complete language has to be infinite, but that it can and it should be decidable, §3.1, and that the complete language has to implement a functional semantics, which is a semantics of syntax, and then it should be part of syntax, §3.2.

¶4 · The thesis that Universal Grammar is Turing complete implies that there are not unlearnable languages, and this together with the fact that natural languages are neither one nor any, requires distinguishing native language, first language, and later languages. Native language is a developed inaccessible mentalese, §4.1, first language is a natural language acquired by genetic endowment and by experience, §4.2, and later languages are those that can be learned, in part by instruction, §4.3. These distinctions define the architecture of human languages, §4.4.

¶5 · Language can be seen as a thinking tool that helps to reason about problems, and then its purpose is to adapt those problems to the hardware that resolves them, §5. From that point of view, the thesis that Universal Grammar is Turing complete has some other implications. One is that language influences thinking, but that a complete language does not limit thinking, supporting the weak version of the linguistic relativity hypothesis. Another is that the thesis explains the rôle played by language in the cognitive gap that separates our species from all others. This is because we are the first and the only Turing complete species, and because Turing completeness is the capacity to do by software whatever hardware does. Then, a single Turing complete living individual can imagine anything evolution could build, so a complete individual can solve quickly what evolution would solve in generations over evolutionary time spams, if ever. Turing completeness is evolutionarily disruptive.

¶6 · A somewhat in reverse order summary is presented as conclusion, §6.

## §2 Thesis

### §2.1 Hierarchy

¶1 · Chomsky (1959) presents a hierarchy of grammars. A *grammar* of a language is a device that is capable of enumerating all the language sentences. And, in this context, *language* is the (usually infinite) set of all the valid syntactic sentences.

¶2 · At the end of SECTION 2 in that paper, page 143, we read: "A type 0 grammar (language) is one that is unrestricted. Type 0 grammars are essentially Turing machines". At the beginning of SECTION 3, same page, we find two theorems.

THEOREM 1. For both grammars and languages, type 0 $\supseteq$ type 1 $\supseteq$ type 2 $\supseteq$ type 3.

THEOREM 2. Every recursively enumerable set of strings is a type 0 language (and conversely).

Then THEOREM 2 is explained: "That is, a grammar of type 0 is a device with the generative power of a Turing machine."

¶3 · From the two theorems we can deduce four corollaries.

COROLLARY 1. The set of all type 0 grammars (languages) is equal to the set of all grammars (languages).

This is because, according to THEOREM 1, type 0 is the superset of all grammars (languages), and more generally because type 0 is unrestricted.

COROLLARY 2. For each Turing machine there is a type 0 grammar (and conversely).

This is equivalent to THEOREM 2, but in terms of grammars (devices) instead of languages (sets).

COROLLARY 3. For each Turing machine there is a grammar (and conversely).

This results by applying COROLLARY 1 to COROLLARY 2.

COROLLARY 4. For each universal Turing machine there is a universal grammar (and conversely).

This is just a special case of COROLLARY 3.

¶4 · The universal Turing machine was defined by Turing (1936). This is the first paragraph of section 6, titled "The universal computing machine":

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

¶5 · Therefore, while type 0 is no condition, so any Turing machine is type 0, and then any table makes a type 0 Turing machine, universality, also known as Turing completeness, is a very strict condition, and not every Turing machine is universal, that is, not every table makes a universal Turing machine. For that reason Turing had to write section 7! And Turing completeness is a very interesting computing property: *Turing completeness* is the capacity to do by software whatever hardware does, because just by writing on the tape of a universal Turing machine we compute whatever any Turing machine computes.

¶6 · Then, according to COROLLARY 4, for each universal Turing machine, which can be programmed to behave as any Turing machine, there is a universal grammar, which can be programmed to behave as any grammar, and conversely, and therefore a universal grammar is equivalent to a universal Turing machine. And remember that Turing completeness is the computing capacity of a universal Turing machine, which is the maximum computing capacity, and then universal grammars are Turing complete.

### §2.2 Universal Grammar

¶1 · In computing, a universal grammar is a device that can be programmed to generate any language, while in linguistics, Universal Grammar is a theoretical concept posited to explain why humans acquire language, but other species do not. So we will write Universal Grammar, capitalized, to refer to the linguistic concept, and universal grammar, all in lower case, to refer to the computing device. Therefore universal grammar is a well-defined mathematical concept, while Universal Grammar is a no so well defined concept in linguistics, and then our next task will be to find a precise definition of Universal

Grammar that we can compare to the mathematical one.

¶2 · According to Chomsky (2005), there are three factors that explain "the growth of language in the individual", that is, the acquisition of language:

- genetic endowment,
- experience, and
- other principles not specific to the faculty of language.

Universal Grammar is what explains why a human child exposed to enough linguistic experience acquires language, and why a chimpanzee does not. Then, Universal Grammar is behind the first factor, genetic endowment, because it is the only factor that makes a difference between humans and apes. And, as genetic code evolves and encodes the body, we can infer two consequences: Universal Grammar evolves, and Universal Grammar encodes our linguistic machinery.

¶3 · That was the evolutionary point of view of language acquisition, but, to compare Universal Grammar with a computing device, we need to see it from the computational point of view. And, if we humans have language, while other species have not, it is because of the specific computational design of our brains. So this is the definition that we will use here: *Universal Grammar* is the human brain circuitry that implements the faculty of language. And then, to characterize Universal Grammar, we have to investigate the linguistic capacity of the human brain. But, before going on, let us see two notes on two assumptions concerning the computational definition.

¶4 · First: The definition of Universal Grammar that we have chosen uses the material assumption. The *material assumption* can be stated like this: when a physical object performs a behavior that provides a function, we assume that the function is somehow instantiated physically in the object. We then say that the object, or a part of it, implements the function. You can consider that the material assumption describes what really happens, or just use the assumption as a figure of speech that allows you to refer metaphorically to a function as if it were a thing.

¶5 · Second: Being computational, the definition of Universal Grammar that we have chosen excludes any non-computational consideration, as for example the optimization of the human vocal tract for language. But, though not explicitly, the original formulation is also excluding those considerations under the assumption that they are not essential for language, because they do not preclude apes from acquiring a full sign language. In this sense, the computational definition, Universal Grammar is the computing circuitry for language, is more explicit and clear than the original one, Universal Grammar is the genetic code for language.

### §2.3 Computing

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

¶2 · Computing is a successful model of problem solving because it abstracts away the limitations of a device in memory and speed from its computing capacity, and because we humans exhibit that computing capacity completely. When Turing wrote his 1936 paper, a computer was a person. So the Turing machine, as it was presented by Turing (1936) himself, models the calculations done by a human computer with a finite internal memory who can access as much external memory as he needs and who has not time limitations. This means that we can compute whatever any Turing machine can compute provided that we can access as much external memory as we need and that we have enough time to accomplish the computation. These two conditions refer to memory access and to available time, and they do not refer to computing capacity, and therefore we are Turing complete in computing capacity.

¶3 · We are Turing complete, and this means that our brain computing capacity is the computing capacity of a universal Turing machine, that is, the maximum computing capacity. Mathematically, being Turing complete is being able to compute any recursive function, and practically a computing device is Turing complete if and only if it can be programmed to perform any algorithm. Then we will call the language needed to program a Turing complete device to perform any algorithm a *complete language*. For example, Turing (1936) used the standard descriptions as complete language, as we saw in §2.1, but other complete languages are possible. The specific requirements that any complete language has to fulfill are presented below, in §3.

¶4 · Turing completeness is the pan-computing capability, because it is the capability of executing any possible computation. A consequence is that any Turing complete device can emulate any computing device, including the Turing complete ones, and this implies that any language, complete or not, can be translated to any complete language. Another consequence is that complete languages are undecidable, just because there are undecidable computations. These are well-known facts in computing.

¶5 · On the other hand, a Turing complete device has to be able to compute any algorithm, and therefore failing just one is disqualifying. Then, that we are the only species that can learn to count up to any number implies that we are the only Turing complete species.

¶6 · And now, using the material assumption again, from the fact that our brain is Turing complete, we assume that there is some circuitry inside our brain that implements Turing completeness. Calling every Turing complete device a *complete engine*, we can say that there is a complete engine inside our brain, but not inside other species brains, to mean that only our species is Turing complete.

### §2.4 Completeness

¶1 · We have seen that our brain capacity is the maximum computing capacity, but here we are only interested in our linguistic capacity. So we must consider the possibility that language does not use the full computing capacity of the brain, and, more precisely, that Universal Grammar does not include the brain circuitry that implements Turing completeness. We will call this possibility the *incomplete hypothesis*. Now we will see that it is wrong, because the opposite possibility is right. That is, we will show that Universal Grammar is Turing complete. But firstly, we will see that the incomplete hypothesis has serious difficulties with evolution.

¶2 · **Evolution** If language does not use the complete engine, as the incomplete hypothesis says, then language could not influence the evolution of the complete engine.

Then the supporters of the incomplete hypothesis should explain, without even mentioning language, why our pan-computing capability, that is, Turing completeness, evolved. Their task is very difficult because computing has a close relationship with language and grammar, as shown by Chomsky (1959).

¶3 · Because mathematics does use the complete engine, for example when dealing with recursion, it would also be very difficult for the supporters of the incomplete hypothesis to explain why two human capabilities, language and mathematics, which are both peculiarities only found in humans, have nevertheless evolved separately.

¶4 · I have argued (in Casares 2017) that syntax and problem solving have co-evolved towards Turing completeness, and therefore that syntax was instrumental in achieving Turing completeness. If this were the case, then evolution would not fit with the incomplete hypothesis.

¶5 · **Argument** If the incomplete hypothesis were right, then language would be unable to use some mathematical expressions. Some will say that this is the case, showing some artificial constructions not found in any natural language. But firstly note that being able to show them, although in quotes, means that they can be used. Secondly see that the artificial languages used in computing also use constructions not found in any natural language, and that, anyway, these artificial constructions can be internalized and used easily, effortless, and unconsciously, by good programmers. And thirdly note that those good programmers and mathematicians use naturally those artificial constructions when they talk to each other.

¶6 · Also, in an English article about a Spanish writer there will usually be some text examples in Spanish between quotation marks. In that case, it would be a syntactic error if the quoted Spanish text would follow the English syntactic rules. Quotation makes a strong case against the incomplete hypothesis, because natural languages can incorporate any foreign or artificial construction by using it.

¶7 · For example, in a Lisp manual written in English there would be some parts following the English rules, and some other quoted parts following the Lisp rules that are being explained in the manual. Lisp is a complete language because we can express and compute any recursive function in Lisp, see McCarthy (1960), and therefore English is a complete language because we can define and mean the whole Lisp in English, see for example the manual by McCarthy et al. (1962), so we can fully translate Lisp to English. This has settled the question, for one needs a Turing complete device to implement a complete language, and then a complete engine is needed to speak English, and, in general, the computing capacity of a universal Turing machine is needed to speak any natural language in which Lisp can be fully explained. The conclusion is that Universal Grammar has to be Turing complete.

¶8 · In fact, any full explanation of recursion in a natural language is a proof that that natural language is complete. For example, Gödel (1930) proves that German is complete, and Turing (1936) proves that English is complete.

¶9 · For these reasons, I disregard the incomplete hypothesis. See that, in the end, this only means that I consider that the Turing complete brain circuitry is part of the circuitry that implements language, or, in other words, that a complete engine is part of Universal Grammar, and therefore that Universal Grammar is Turing complete, or, in other words, that Universal Grammar is a universal grammar.

## §3 Requirements for completion

### §3.1 Syntax

¶1 · That Universal Grammar is Turing complete seems to contradict the linguistic consensus that, in the hierarchy of Chomsky (1959), natural languages are located between context-free and context-sensitive languages, see for example Stabler (2014). To see that there is not contradiction, you have to see that the syntax of a complete language can be decidable. And this is easy; take for example Lisp as presented by McCarthy (1960): Lisp syntax is context-free, and Lisp itself is complete. That's it.

¶2 · When the syntax of a language is decidable and the language is complete, and then undecidable, we have to blame semantics for the undecidability, and for the completeness! But take Lisp again. Lisp is run by computers, and a computer is a syntactic machine; in fact, the computer is the prototype of syntactic machine, because it is devoid of meanings, isn't it? Well, yes and no. Let me explain.

¶3 · First, remember that a Turing complete device can perform any computation that is entered to it as a program written in its complete language. Then, it is a requirement that any computation could be expressed in the complete language. And it is also a requirement that the meaning of any such program be known by the Turing complete device to perform the computation specified. The first requirement is for syntax, and the second for semantics. Let us see them one by one.

¶4 · The number of computations is equal to the number of Turing machines, that is equal to the number of natural numbers, as proved by Turing (1936), where the set of natural numbers $\mathbb{N}$ is an infinite enumerable set. This means that any infinite enumerable set can be used as syntax of a complete language, because then we can map each possible computation to a different syntactic object. Thus, any computation can be expressed in the complete language, and we can already state the syntactic requirement for completion: the set of syntactic objects has to be infinite enumerable, or bigger. In mathematical terms, $|S_c| \geq \aleph_0$, where $S_c$ is the set of the syntactic objects of a complete language, and $\aleph_0$ is the smallest infinite cardinal number, which is also the cardinality of the natural numbers, $\aleph_0 = |\mathbb{N}|$.

¶5 · Remember that an infinite enumerable set, if it is recursive as defined by Post (1944), then it is decidable. This shows that the syntax of a complete language can be decidable. Then, while mathematics states the syntactic requirement for completion, 'the set of syntactic objects has to be infinite', engineering states the syntactic recommendation for completion: the set of syntactic objects should be infinite decidable. For example, natural language, Lisp, Turing's standard descriptions, and even arithmetic, all fulfill the syntactic requirement for completion, and all follow the recommendation.

### §3.2 Functional semantics

¶1 · Any infinite set can be the syntax of a complete language, because then we can express any computation in the language. This is nice, but still useless. The Turing complete device has to take any syntactic object expressing a computation and effectively perform that specific computation. This is the semantic requirement for completion, because the Turing complete device has to know the meaning of the syntactic object to perform the calculation.

¶2 · The semantic requirement for completion can be formulated in different ways. When the infinite set of syntactic objects is the set of natural numbers, $\mathbb{N}$, which is the canonical

infinite enumerable set, we get the base case. In the base case, the semantic requirement for completion is that the Turing complete device can be programmed to calculate any recursive function of natural numbers. Then, in the general case, the semantic requirement for completion is that the Turing complete device can be programmed to calculate any computable function of its syntactic objects. Because of this, we will call any semantics needed to fulfill the semantic requirement for completion a *functional semantics*. Please note that here we will only use 'functional semantics' in this sense.

¶3 · For the case of natural language, the closest is the case of lambda-calculus. In the case of lambda-calculus, which is simpler than Lisp and uses the same syntax, the semantic requirement for completion is basically fulfilled by an operation called beta-reduction and a terminal symbol named lambda, see for example Barendregt (1985). Lambda tags the variable in a function definition, and beta-reduction is a kind substitution that can be applied to variables and to composed syntactic objects, and that obey some scopes of application. As any meaning can be assigned to a variable, then variables have to be processed independently of their meanings, and therefore a variable is like a pronoun, or a word without meaning. By the way, calculating with words independently of their meanings is a requirement for full problem solving, where pronouns as 'what' and 'it', or temporal words introduced by 'let-us-call-it-x' formulas, do refer to something yet unknown, and even to nothing if eventually the problem has not any solution.

¶4 · In summary, in the case of lambda-calculus, functional semantics deals with terminal symbols, variables, and scoped substitutions of variables and syntactic objects. This can be translated easily to natural language: terminal symbols are function words in general, variables are pronouns, and scoped substitutions of pronouns and syntactic objects are operations routinely used by natural language to resolve references before reaching the meanings of content words. If Universal Grammar is Turing complete, as we propose, then Universal Grammar has to include the computing resources that implement a functional semantics, and implementing one like this of lambda-calculus should not be controversial, because function words natural place is not the lexicon, and the substitutions mentioned can be executed by a computer that has no content words. In any case, functional semantics is a semantics of syntactic terms and syntactic operations that deal with computations of words independently of their meanings, so functional semantics is the semantics of syntax, and then it should be part of syntax. We will call a syntax extended with a functional semantics a *complete syntax*.

¶5 · In an extended syntax, the corresponding extended syntactic object has to include completely the scope of the substitutions. Then, for example, when the reference of a pronoun oversteps the sentence boundaries, the extended syntactic object will be multisentential. This means that even a language with a very simple syntax, as for example the Pirahã language described by Everett (2008), can be complete.

¶6 · My conclusion is that Universal Grammar satisfies both requirements for completion: it uses an infinite decidable set of binary trees for its syntactic objects, and it implements a functional semantics on them. The binary trees can be generated by Merge, as proposed by Berwick & Chomsky (2016), but the implementation of a functional semantics in the syntax core, which is what I am calling Universal Grammar, is beyond their proposals. The functional semantics of Universal Grammar would include, at least, scoped substitutions, temporal word definitions, and conditionals; see Casares (2017).

# §4 Learnability

## §4.1 Native language

¶1 · Our thesis that Universal Grammar is a universal grammar implies that there are not unlearnable languages. This is because any Turing complete device can emulate any computing device, including the Turing complete ones, so any language can be translated to any complete language, as we saw in §2.3.

¶2 · But some clarifications, which require an additional definition, are in order. We will call the complete language in which a Turing complete device is directly programmed the *native language* of the device. For example, in computing the native language is called machine language or machine code, because it is executed directly by the computer hardware. If the machine language is complete, then we can implement any language on it, let us call it high-level language, and again, if the high-level language is complete, then we can implement any language on it, and so on and on; more on this in §4.4.

| | |
|---|---|
| High-level language | Software |
| Machine language | Hardware |

Then, the first clarification: any complete language can be translated to any complete language, but only a Turing complete device can do the translation, and the Turing complete device needs a native language, which is complete. Only once the native language is built in the hardware, are all complete languages fully translatable.

¶3 · In the case of persons, our assumption here is that our native language cannot be used directly for communication. The reason is that if our native language could be used directly for communication, needing no adapter, then it would be used and all natural language complete syntaxes would be the same, but they are not.

## §4.2 First language

¶1 · Another general agreement in linguistics is that not every language can be a natural language, and for example Berwick & Chomsky (2016), page 126, affirm that there are not natural counting languages. If this is true, then some additional restrictions are imposed during the critical period, possibly to make language acquisition easier, because counting languages can be learned. For example Fortran, one of the most successful computing languages, was a counting language for a long time, from its introduction in 1956, see Fortran (1956) pages 7–8, and continuing for decades, see Fortran (1966) §3.2–3.4, and Fortran (1977) §3.2–3.4, until 1990 when free-form source input was allowed, see Fortran (1990) §3.3.

¶2 · The native language is necessarily one, because it is the language in which the Turing complete device is directly programmed, and we can learn any language, because we are Turing complete, but it seems that human natural language is neither one nor any. Therefore, in persons we should distinguish native language, first language, and later languages. Let me present you a sketch of the situation.

¶3 · The native language is complete, genetically coded, and hardwired in our brain. Possibly, our native language evolved for thinking, specifically for problem solving, and it takes advantage of the natural parallelism of the neural network that is the brain. As it is, the native language cannot be used for communication, possibly because communication requires serialization, and then an interface with the sensorimotor system needs to be

enabled. That interface is genetically coded, at least partially, and it seems that it has to be developed during a critical period. Once the interface is developed, the native language adapted by the sensorimotor interface results in the first language. The first language can be complete, and it should be complete or otherwise it could not express and mean any possible recursive function, that is, any possible algorithmic rule for change. All human natural languages are first languages, and here we will assume that all are complete. Thereafter, we can use the first language to learn by instruction any other language, natural or artificial, complete or not. Because of the nature of the brain processes, instruction should be supplemented by use and practice in order to convert the learned language into hardware.

¶4 · In summary, we should distinguish native language development, first language acquisition, and later languages learning.

| | | |
|---|---|---|
| Later languages | ∘ | Learning |
| First language | ∘ | Acquisition |
| Native language | ∘ | Development |

¶5 · The question now is: How do these distinctions affect Universal Grammar? To answer it, we must go back to the fundamentals: Universal Grammar is the genetic code for language. Switching again from the code to the coded computing machinery, the native language is implemented by Universal Grammar, and later languages are not. In the case of the first language, the part of the interface that is genetically coded, which are the conditions that were being enforced during the critical period, is implemented by Universal Grammar, and the part acquired by experience is not. As I think it would be beneficial to keep native and first language issues separated, we could say that the core Universal Grammar implements our native language, while Universal Grammar generally implements the base for all natural languages, including the native language and the linguistic conditions imposed for acquisition; see below §4.4.

### §4.3 Later languages

¶1 · By instruction given in the first language, we can learn any other complete language, as Spanish or Lisp. That is the reason why there are teachers and manuals to learn Spanish or Lisp. Instruction is a kind of learning available only to Turing complete devices, because it requires a complete language. Note that 'instructed' for persons corresponds to 'programmed' in computing. Then, the second clarification: once we have acquired our first language, then we can also learn by instruction.

¶2 · With this distinction between language development, acquisition and learning, we should rephrase the belief by Chomsky (1988), page 149, that Universal Grammar implies that there are "unlearnable" languages. He argues that Universal Grammar has certain definite properties, which are shared by all natural languages, and that these same properties exclude other possible languages as "unlearnable", meaning that we can not "learn" a language that does not have these properties. As this applies to first language acquisition, we should instead say that Universal Grammar implies that there are unacquirable languages. So now we can state the third clarification: not every language can be acquired, but we can nevertheless learn any language.

¶3 · The same correction should be applied to Gold (1967), if we hold the distinction. He models two ways of "learning" a language: from text, that is, from a set of just

positive examples; or from an informant, that is, from a set of evaluated (as right or wrong) examples. This can be sufficient for acquiring a language, but it is insufficient for learning a language, because then there is a third way: instruction.

### §4.4 Language stacks

¶1 · In computing the stack of languages can also include a very high-level language on top, an intermediate language in between, and even microcode below.

| | |
|---|---|
| Very high-level language | |
| High-level language | |
| Intermediate language | Software |
| Machine code | Hardware |
| Microcode | |

Again, if all the languages in the stack are complete, then we can still program any algorithm on the upper language, and it will be properly translated down to the hardware to be executed. But technically, when microcode is used, the native language of the computing device is the microcode, instead of the machine code, even though microcode is not accessible from outside, because the device only accepts machine code instructions, which are translated by hardware to microcode in order to be executed.

¶2 · Note that in persons, microcode would correspond to an inaccessible mental language, known as mentalese, and machine code would correspond to our accessible natural language. Then, using this analogy, our native language would be our mentalese and our first language would be our machine language. But these are just computing approximations that assume a fixed hardware, while the neural hardware of a living individual changes, and therefore the actual situation will be fuzzier.

¶3 · In the case of the language used in this paper, the stack of complete languages in which it is written is like this.

| | |
|---|---|
| Mathematics | Learned |
| English | Learned |
| Spanish | Acquired by experience |
| Natural language | Acquired by genetics |
| Mentalese | Developed |

My first language is Spanish, and as any first language, part was acquired because of first factor causes, genetic endowment, and part was acquired because of second factor causes, experience, see §2.2. The part that depends on experience is, basically, the lexicon and the complete syntax tuning. Then, Universal Grammar is below the horizontal rule.

## §5 Relativity and evolution

¶1 · Usually, in computing, an algorithm is coded in a high-level language, as Lisp, and then translated to the machine language of the computer by a compiler or an interpreter. We will assume that the efficiency of a computation is inversely related to the amount of computer memory used and to the number of clock cycles needed to complete the execution of the algorithm. Then, the efficiency of a Lisp computation depends on the number of Lisp clauses needed to code the algorithm, which depends on how much the problem to be solved by the algorithm fits the peculiarities of Lisp, and on the number of machine instructions that the Lisp clauses generate, which depends on how much Lisp fits the peculiarities of the machine language of the computer.

¶2 · The conclusion is that the language used to express an algorithm that solves a problem affects its computation, although if the computer is Turing complete, then any algorithm can be computed. Correspondingly for persons, language influences thinking but, if it is complete, then language does not limit thinking, which is the weak version of the linguistic relativity hypothesis.

¶3 · While we can program any algorithm in any complete language to solve a problem, the difficulty of programming it and the efficiency of the resulting program depend on how much the problem fits the complete language and on how efficient is the implementation of the complete language in the Turing complete device. From this point of view, language is a thinking tool that helps to express and to reason about problems, and then its purpose is to translate those problems to the hardware that resolves them. The implication for evolution is that Universal Grammar is an adaptation to the problems that humans were facing and to the peculiarities of their neural hardware.

¶4 · That Universal Grammar has not evolved recently is shown by the fact that "an infant from a Papua New Guinea tribe without any human contact for 60,000 years, growing up in Boston from birth," would acquire the very same Bostonian English that a neighbour local child speaks, an example taken from page 150 of Berwick & Chomsky (2016). To me, this happens because Turing completeness is a natural computing concept, and then achieving it is reaching a peak of fitness, and also the converse.

¶5 · Because Turing completeness is a pure computational property that does not require any anatomical modification, I think it was achieved in the time frame of the anatomically modern *Homo sapiens*, but before the African exodus. So I date the achievement of Turing completeness in the same time frame that Berwick & Chomsky (2016) date the emergence of Merge, from 60k to 200k years ago.

¶6 · But while the computational value of Merge is limited, Turing completeness is a well-defined systemic computational property that makes a difference. In informal terms, Turing completeness causes the gap that goes from a non-programmable calculator to a full-programmable computer. More formally, Turing completeness is the capacity to compute any recursive function entered as data, which is the capacity to calculate any algorithmic rule for change. In other words, Turing completeness is the ability to do by software whatever hardware does. A consequence is that a Turing complete individual, as you and me, can imagine any change and its effects in his head before executing it, while evolution has to modify the species body just to apply its trial and error procedure. Thus, Turing completeness is evolutionarily disruptive. And, that our species is the only Turing complete one explains the cognitive gap between our species and the rest.

## §6 Conclusion

¶1 · Computing is a point of view that defines decidability and Turing completeness, and that distinguishes program from data and hardware from software. These concepts are needed to explain the cognitive gap between our species, *Homo sapiens*, and all others: we are the first and only species that achieved Turing completeness. Being Turing complete means that we can do by software whatever hardware does, so we can imagine anything that evolution could build, thus speeding up problem solving enormously. Turing completeness needs a complete language in which programs are treated as data, and any complete language has to fulfill two requirements: its syntax has to be an infinite set, which can be decidable, and it has to implement a functional semantics. Functional semantics is the semantics of syntax, and then its place is the core syntax. This way we reach our thesis: Universal Grammar is Turing complete.

¶2 · Our thesis that Universal Grammar is Turing complete implies that there are not unlearnable languages. This together with the fact that natural languages are neither one nor any, demands a distinction between native language, which is developed, first language, which is acquired, and later languages, which are learned, where all natural languages are first languages, and our native language is not accessible. Some consequences follow from these distinctions:

- Not every language can be acquired, but we can learn any language.
- Instruction is a way of learning, once we have acquired a first complete language.

¶3 · Computing is a problem solving point of view from which language is a tool to translate problems to the hardware that resolves them. Some more consequences follow:

- Language influences thinking, but a complete language does not limit thinking.
- Turing completeness is a fixed point in the evolution of Universal Grammar.

Universal Grammar evolved as a compromise that took into account its various functions and its implementation. Universal Grammar functions include, at least, language, mathematics, and problem solving, and its implementation had to resolve its interaction with other brain modules while still satisfying other engineering conditions.

¶4 · This is the theory I am proposing here for your consideration. The theory is firmly founded on two bases: firstly, that the best point of view to explain syntax and language is computing, which is the essence of Chomsky (1959), and secondly, that from the point of view of computing, we humans are Turing complete, which is the essence of Turing (1936). Then there are some assumptions that you will have to assess, but they are very few, and the only important thesis is that Universal Grammar is Turing complete.

## References


Barendregt (1985): Henk P. Barendregt, *The Lambda Calculus, its Syntax and Semantics*; Revised Edition, Studies in Logic and the Foundations of Mathematics, Vol. 103, North-Holland Publishing Co, Amsterdam, 1985, ISBN: 0-444-87508-5.

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

Casares (2017): Ramón Casares, "Syntax Evolution: Problems and Recursion"; DOI: 10.6084/m9.figshare.4956359, arXiv:1508.03040.

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.

Chomsky (1988): Noam Chomsky, *Language and Problems of Knowledge: The Managua Lectures*, Current Studies in Linguistics, 16; The MIT Press, Cambridge MA, 1988, ISBN: 978-0-262-53070-5.

Chomsky (2005): Noam Chomsky, "Three Factors in Language Design"; in *Linguistic Inquiry*, vol. 36, no. 1, pp. 1–22, Winter 2005, DOI: 10.1162/0024389052993655.

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.

Everett (2008): Daniel L. Everett, *Don't Sleep, There Are Snakes: Life and Language in the Amazonian Jungle*; Vintage, New York, 2008, ISBN: 978-0-307-38612-0.

Fortran (1956): International Business Machines Corporation, "The FORTRAN Automatic Coding System for the IBM 704 EDPM"; International Business Machines Corporation, New York, Programmer's Reference Manual, October 15, 1956.

Fortran (1966): American National Standards Institute, "FORTRAN: USAS X3.9-1966"; United States of America Standards Institute, New York, NY, 1966.

Fortran (1977): International Organization for Standardization, "FORTRAN: ISO 1539:1980"; https://www.iso.org/standard/6127.html.

Fortran (1990): International Organization for Standardization, "Fortran: ISO/IEC 1539:1991"; https://www.iso.org/standard/17366.html.

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).

Gold (1967): Mark Gold, "Language Identification in the Limit"; in *Information and Control*, Volume 10, Issue 5, pp. 447–474, May 1967, DOI: 10.1016/S0019-9958(67)91165-5.

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

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

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.

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.