Grammar is NOT a Computer of the Human Mind/Brain

Prakash Mondal

Abstract

This paper will attempt to debunk the idea that human language grammar as part of the Faculty of Language (FoL) is intrinsically a computing device. The central argument here is that grammar does not compute. One way of demonstrating this is to show that the operations of grammar in the Generative model do not have the character typical of computations. Thus, the central operation of grammar Merge, which combines lexical items to produce larger expressions, can be defined as a recursive function, but it does not share the inductive properties of recursive functions in mathematics in view of the consideration that recursive functions define computability. On the other hand, if the language faculty is a computing system, the language faculty must inherit the halting problem as well. It is easy to impose the halting problem on the selection of lexical items from the lexicon in such a manner that FoL may or may not terminate over the selection of lexical items. We can say: there is no FoL way of telling if FoL will ever terminate on *x* or not when *x* is a selection from the lexicon. The halting problem for FoL is disastrous for the view that grammar is a computing system of the brain/mind since it detracts from the deterministic character of FoL. This has significant repercussions not just for grammar that cannot be restricted to any limited view of mental computation but also for the nature of the cognitive system as a whole since any cognitive domain that is (supposed to be) language-like cannot be said to compute as well.

Keywords

Grammar, Faculty of Language, Computation, Merge, Halting Problem, Cognition

Introduction

The idea that the grammar of human language is a formal system which forms part of the human mind realized as the Faculty of Language (FoL), and is a computing device has been advanced and popularized by the influential paradigm of Generative Grammar. The reason why this has been highly influential is that human language grammar is thought to be a kind of software installed on the brain hardware when a language is acquired by members of *Homo Sapiens*. Human language grammar, as part of our mind, which seems to be a non-physical intangible entity, is positioned at a higher-level of abstraction than the brain structures in which it is ultimately realized, just like the level of software in any modern day computer is situated at a level of abstraction removed from the level of the electrical circuitry. This makes the abstractness of the system of grammar

possible. Thus, this is supposed to explain why grammar as a formal system, just like a mathematical system (for instance, the system of arithmetic), generates an infinite number of linguistic expressions from a finite stock of words. While many theoretical linguists and cognitive scientists have, over the period, criticized and contested another influential view associated with the theory of Generative Grammar (Chomsky 1995, 2000), which is that humans are born with a biologically-hardwired capacity for language called Universal Grammar. Theoretical linguists and cognitive scientists have not found anything wrong with the idea that the language faculty, which is supposed to be a mental system, is *intrinsically* a computing device.

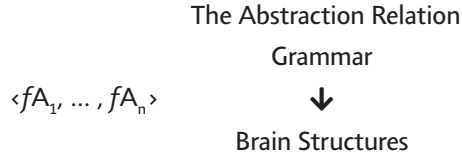

Figure 1: The Abstraction Relation

Here, grammar can be thought to abstract away from the underlying brain structures by way of a number of abstraction functions such as $fA_1 \ldots fA_n$, which perform a series of mappings to move away from the neurobiological structures and then reach into the higher level of abstraction from the lower-level of realization in the brain hardware. This ensures that grammar remains at a distance sufficient for a level of abstraction removed from the level of physical realization. Thus, for example, if there is an abstraction function fA_i that maps some neural state onto some state of the FoL, the idea is that if some state of the FoL changes to another state, then the designated change must be caused in some way by a change in the underlying brain state. There must be many such abstraction functions, since states of the FoL are abstractions whose configurations and transitions are to be related to the state changes in the neural hardware in ways that can be formally characterized. In this way, the state transitions in the physical system of the brain are to be mapped onto the state transitions of the FoL. This ensures that grammar, as part of the FoL, can execute operations that are sufficiently abstract, on the one hand, and are yet physically realized in the brain structures on the other. But, these operations are customarily understood and claimed to be computations, and grammar as a system performing such computations is thus a computing device. This is so because the FoL is supposed to be a computational system. Hence, before we proceed further, we need to

understand what computation is and, in what sense, grammar, or for that matter, the FoL is said to be a computational system.

In what Sense is Grammar (FoL) a Computing Device?

When one raises a question about whether something is computational, a lot depends on whether one deploys the right concept of computation to a phenomenon at hand, while checking if the given phenomenon falls under computation. The concerns are similar when we focus on language and wonder, in what sense, the system of grammar performs computations. Even though there is an amount of vagueness embracing the notion of computation applying to the FoL, it appears that linguistic computation, as performed by the FoL, jells well with the classical view of computation on which symbolic inputs are mapped to symbolic outputs according to some well-defined rules that are defined over those symbolic units which exist in the form of linguistic strings. This notion of computation lies in the narrowest region in the whole hierarchy of varieties of digital computation (Piccinini and Scarantino 2011). This notion of linguistic computation has been adopted in much of formal linguistics implicitly or explicitly because the representational vehicles of language are discrete in form. But, it is not clear whether we can take linguistic computation to be a kind of *generic* computation covering both digital and analog computations. Although this question may not a have clear answer, as most linguistic frameworks that apply the notion of computation do not demarcate the notion of computation, the answer is more likely to be no. What is noteworthy here is that the digital notion of computation has been the central principle in the model of cognition advanced in the field of cognitive science, in general, and presupposed in much of theoretical linguistics, in particular. The analog sense of computation is not in harmony with the view of linguistic computation on the grounds that analog computational processes are driven and determined by the *intrinsic* representational content of the vehicles that are analog in nature, whereas digital computation involves mappings of inputs onto outputs that are executed without any regard to the content-determining properties of the representational digital vehicles (O'Brien and Opie 2011). This squares well with what Chomsky (1980) has espoused, especially when he thinks of linguistic computation in the sense that the mentally instantiated system that can be identified with the FoL is a computing device of some sort. Thus, he maintains that '… "autonomous" principles of mental computation do not reflect in any simple way the properties of the phonetic or semantic "substance" or contingencies of language use.' From this, we can draw the conclusion that linguistic computation does not plausibly

encompass the analog sense of computation. Taken in the light of these considerations, linguistic computation cannot be regarded as a type of *generic* computation that embraces both digital and analog computation.

As we proceed, we shall require the appropriate notion of computation to accompany the critique that follows. This will help get a grip on the concept of linguistic computation when dealing with the view that grammar is a computing device of some sort. Central to any notion of computation are the following important elements: (i) a function that is computed; (ii) a system which computes the function; and, (iii) an effective procedure (also called an *algorithm*). This is also evident in the *Church-Turing Thesis*, which states that anything that can be computed with an effective procedure in the physical world can be computed in Turing machines. Related to this is the computational thesis in mainstream cognitive science that postulates that the human mind, or the brain, is itself a computing system. The rationale seems to make sense only when we suppose that the human brain is ultimately a physical object of some kind that can run computations just like any other physical system that executes computations. We may now wonder what the right physical system is that runs, or is appropriate for, linguistic computations. If we follow the lines of thinking adopted in Chomsky (1995, 2000), then we can make sense of the physical system fit for linguistic computation. In fact, the physical system for running linguistic computations is a system or module of the brain dedicated to language. Thus, it is the language faculty in the brain/mind that computes because the language faculty is conceived of as the physical organ specific for language situated within the confinements of our brain. The language faculty computes because it is supposed to have a *computational procedure* that engages in all kinds of linguistic computation. The next essential ingredient of computation is a function, or rather a computable function. The domain of such functions in formal linguistics may well correspond to the domain of formal operations that apply to structures to make structural distinctions of linguistic representations, when linguistic structures are inserted, erased and thereby altered. In this sense, we can take linguistic computations of the FoL to be the operations of a version of the Turing machine that works on a potentially infinite tape and reads, writes, or erases symbols on the tape. In a nutshell, the functions that can fall under linguistic computation are those which subscribe to the formation, substitution, and deletion of phrases, sentences, or larger linguistic expressions. Significantly, this is the *process-oriented* aspect of computation implicit in the specification of the Turing machine. That the system of grammar can be taken to be executing computable functions, or rather algorithms, accords well with the *abstraction-oriented* aspect of computation, which consists in the specification of computable functions that can be implemented by

algorithms in a system. If the computational character of the FoL is considered under the cover of the abstraction-oriented aspect of computation, then all that really matters is the specification of computable functions to be implemented in the putative computational system of the language faculty. That the specification of computable functions that grammar, as a system, is supposed to execute is possible has been shown for the central operations of grammar in the Minimalist model of grammar, especially for Merge, which is understood to concatenate syntactic objects (see Mondal 2014). In this context, Foster's (1992) notion of an algorithm as a sequence of transitions between the states of a machine is handy enough. In particular, Merge combines two syntactic objects to form a single syntactic object (which is actually a set). Thus, for a sentence like 'John loves a car', we have Merge (a, car) = {a, car} and Merge (loves, Merge(a, car))= {loves, {a, car}} and then (John, Merge (loves, Merge(a, car)))= {John, {loves, {a, car}}}. Note that the formulation of Merge for the generation of the linguistic expression 'John loves a car' is recursively specified--more will be said on this below. An algorithmic representation of the operations of Merge for the phrase 'a car' can thus be schematized as (1), by following Foster.

(1) $[SO_1$: a SO_2: car L: Σ] → [SO: a, car L: Σ] → [SO: {a, car} L: Σ] →

[SO: Σ {a, car}]

Here, SO is a syntactic object and L denotes the label of an SO, and Σ is the actual value of L. Thus, each item on the left of the colon is the label, and the one on the right designates the value of that label. Each item enclosed within braces represents a 'snapshot' of the state of a computation, and the arrow represents a transition between one such state and another. Now that the operation Merge has been shown to have a computational character in defining computable functions that can be coded as algorithms, the core generative engine of the FoL can be said to run computations by virtue of containing computable functions defined over the symbols the system of grammar operates on. This ensures that the system of grammar is viewed as a computing device whose computational nature can be characterized in the standard terms of the execution of computable functions.

Why Grammar (or FoL) is not a Computing Device

We may now look into the reasons why grammar or the FoL cannot be considered to be a computing device. Since the system of grammar is a computing device in virtue of defining computable functions, it possesses the abstract capacity of generating infinite

linguistic expressions *just as* an abstract system of arithmetic generates infinite arithmetic expressions. The relevant mathematical property here is the recursive character of the operations the system of grammar instantiates. An analogy from mathematics can be drawn in order to demonstrate how grammar as a system is *intrinsically* computational. Take, for instance, a recursive function that increments the value of a given number by 1: f(n)= n+1 when *n* is a natural number. Thus, f(5)= 5+1= 6 when *n*=5, for example. This function is recursively defined, in the sense that the given function is specified in terms of each calculated value of its own output. Hence, this function can also be specified in a manner that involves the invocation of the same function. So, we can write f(n)= f(n-1) +1. It may be noted that an inductive definition forms an intrinsic part of the formulation of the incremental function here. This is because the inductive definition licenses the inference that the function can be specified in terms of each calculated value of its own output by way of an invocation of itself. As shown above, the generative mechanism of grammar has a recursive characterization in virtue of the fact that the generation of an infinite number of linguistic expressions is part of the recursive definition of the operation Merge. That is, the putative computational system of the language faculty possesses this mechanism by virtue of having the operation called Merge. Therefore, it seems clear that all that matters is the specification of the function concerned, not how this function is implemented in the language faculty *in real time*. This must be so because Merge is defined as a function *in intension*. If grammar is a computing system in this sense (as far as the mapping function so defined is concerned), it is not unreasonable to think that the relevant properties of recursive functions that hold true for the set of natural numbers should also be found in the set of natural language expressions generated by Merge or by any conceivably analogous computational mechanism of grammar. Let's see how we can test this formal parallelism. Suppose we have the following sentences which are output by Merge:

(2) (Amy + (trusts+ (a + man + … + … +…)))

(3) (Amy +…+ … + (trusts+ (a + man)))

The sentence in (2) can be taken to have an unbounded expansion which goes on like this: 'Amy trusts a man who is known to have three mansions which are located in three different countries that form a certain contour around a place that defies any description …'. Likewise, (3) can be also be unboundedly long such that its expansion may run like: 'Amy who is one of the finest scholars at our university which motivates the study of culture in unexplored territories which may not have any access to education… trusts a

man'. The problem for Merge is that it cannot get off the ground in (2) since Merge, as a constructive operation, starts and continues to work in a bottom-up fashion, whereas it can never terminate, even if it does start in (3). Note that recursively defined functions in mathematics are such that they may never terminate, hence this particular argument certainly cannot have the appropriate force it should have, since, after all, Merge taken as a mapping function is defined in intension. However, functions operating on (the set of) natural numbers, as in $f(n)= n+1$, *at least* get off the ground when there are inputs to be mapped onto outputs, regardless of whether they terminate. To put it in other words, functions operating on (the set of) natural numbers do not spell out the problem of not starting in the first place, while Merge contains the germ of the problem of not starting in the first place as well as inheriting the problem of non-termination. One may try to circumvent this problem for Merge by postulating null items that are *assumed* to exist in the unboundedly long sentence in (2) in order to save Merge from getting stuck into this trap. If the same strategy is adopted, then the null items which may be taken to be the stand-ins or proxies for the relative clauses constituting the expansion in (3) may also be assumed to have been Merged. For all its appeal, this strategy is groundless because items empty of substance are inserted in a linguistic expression which is not even a well-formed expression and perhaps does not even exist due to its unbounded or unfinished form. We end up inserting items empty of substance into an expression, which, as a whole, is already empty of content. The result is anything but a meaningful statement. Plus, this detracts from the operational character of Merge because Merge does not concatenate null items. This is the case in the Minimalist model of the language faculty, for there is a ban imposed on the FoL that disallows items which have not present in the selected set of lexical items on which computations are to operate. Besides, null items for chunks as big as relative clauses cannot be selected from the lexicon, nor can they be justified on linguistic grounds, since nothing would then prevent one from postulating null sentences whether simple or complex.

The worry does not, of course, stop here. There is another, deeper, more fundamental problem residing in the postulation of formal parallels between recursive functions in mathematics and the putative computational mechanism of grammar. Just for instance, the principle of *mathematical induction* applies to all well-formed functions when it is used as a proof technique to test whether something holds for an infinite set because we cannot check all items in a potentially infinite set. So, as per the principle of mathematical induction, if some proposition P holds for n, it also holds for $n + 1$. The second step in this formulation constitutes an *inductive* generalization that may also be aligned with various other kinds of generalizations drawn inductively by human beings. Let's now reconsider

the example in (2) to determine whether mathematical induction can be applied to it. The example (2) has been represented the following way in (4).

(4) 'Amy trusts a man' + Rcl^k (where $Rcl^k = k$ number of relative clauses)

Since it is necessary to render (2) in a manner that makes it amenable to the application of mathematical induction, the formulation of (2) in (4) serves to demarcate the domain, that is, the portion Rcl^k, over which mathematical induction can be taken to apply. One of way accomplishing this is the following way of characterizing the relevant set so that we state that mathematical induction applies over the set in (5).

(5) {'Amy trusts a man', 'Amy trusts a man who is known to have three mansions',

'Amy trusts a man who is known to have three mansions which are located in three different countries' …}

But what are the appropriate properties of this set, or of the members of this set, that can help establish that some proposition precisely formulated holds for the *n+1*th expression only if it holds for the *n*th expression? In what sense can the expression 'Amy trusts a man' be supposed to be the *n*th expression? Or, in what sense can the expression 'Amy trusts a man who is known to have three mansions' be the *n+1*th expression and so on? What are the exact properties of these expressions such that their succession can mimic that of natural numbers when the natural numbers that are inputs or outputs of a function are defined in terms of a function? One suggestion that can be implemented here is that the relevant proposition that needs to be tested has to be formulated by tracking the *depth* of concatenation of relative clauses. That is, one may say that 'Amy trusts a man' is an expression with the value of the depth of concatenation fixed at 0, and similarly, 'Amy trusts a man who is known to have three mansions' has the depth of concatenation set at 1 and so on. This may be supposed to reflect the progression of these expressions at par with that of natural numbers. So the proposition to be tested is that the concatenation of a relative clause to a sentence whose verb phrase is transitive returns a well-formed expression of English. This can be couched in terms that may be supposed to ride especially on the inductive generalization that the attachment of a relative clause to a sentence whose verb phrase is transitive *always* yields a well-formed expression of English. The specific rule may be formulated in terms of *phrase-structure rules* familiar in formal linguistics. The advance of formulating such a rule is necessitated by the consideration that the rule has to be maximally general so that inductive definitions hold

true for (4) or even (5). Let formulation in (6) be the exact phrase-structure rule that we need to capture this inductive definition:

(6) Sentence (S) → Noun Phrase (NP) Transitive Verb Phrase (TVP) + Rcl^{k}

But the problem is that the rule in (6) can never ground (4) or (5) in an inductive generalization, simply because rules like this overgenerate. Nothing stops (6) from generating (7), below:

(7) *Amy trusts a man which is known to have three mansions which are located in oneself that forms a certain contour around hers that has any description …

Likewise, there is nothing that can prevent one from having an expansion in (4) at some *n+1* level that renders the whole expression grammatically illegitimate, as in (8), below:

(8) *Amy trusts a man who is known to have three mansions which will sold tomorrow.

An attempt to import grammatically relevant, context-sensitive information, and selectional properties of predicates and other expressions into the contexts of (7-8) will inevitably vitiate the prospect of having a rule that will possess such a general character as to be amenable to an inductive definition. This is because when we say that if the incremental property of the function *f*(n)= n+1 considered above holds for the number *n*, induction guarantees that it will also hold true of the *n+1*th number. That is, mathematical induction ensures and safeguards the *generality* of the induction without any provisos or conditions fixed for the induction to apply in the first place. Needless to say, this is doomed to fail for natural language expressions. There is the following dilemma when we turn to natural language. On the one hand, we require something like a function that can have the desired formal generality across a potentially infinite range of expressions, and on the other hand, the nature of natural language grammar is such that it defies the formulation of any such function. It is important to recognize, in this connection, that neither the compositional function nor the intuitive sense of concatenation can serve this purpose. The former is of no substantive value *in this particular case* because natural language abounds in non-compositionally formed expressions (idioms, for example). Thus, we can have expressions such as 'take for granted,' 'beat around the bush,' 'call time on,' etc. whose meanings are not strictly determined by the combinations of the meanings of the parts of the whole expressions. Concatenation, on the other hand, as

an operation is too trivial to have any linguistic value since the output expressions from the operation of concatenation can be deviant or ungrammatical. There is nothing that can, for instance, prevent one from concatenating 'an' with 'ball' or even 'for' with 'done', which will yield 'an ball' or 'for done,' both of which are ill-formed in English.

Beyond that, Merge cannot defined as a recursive function, given that recursive functions define computability. Thus, for example, addition is a recursive function because it can invoke itself as an input. So +(3, 5)=8 and then +(8, 8)=16 can be better expressed as +(+(3,5), 8)=16. Also, note that the inputs and outputs are all members of the set of natural numbers. This is not so for natural language. If Merge(John, runs)= John runs, we cannot have something like Merge(Merge(John, runs), John runs)=John runs John runs. That is, 'John runs John runs' is not a well-formed string in English. This can be generalized to any language other than English. The relevant property is called the closure property of functions or operations defined on natural numbers. Closure properties make it possible for natural numbers to be defined within the bounds delimited by the set of natural numbers. That is, it is closure properties of natural numbers that tell us that both 5 and 4 in 5+4 =9 are natural numbers and so is the number 9. Similarly, both the input numbers and the output number involved in the operation of multiplication in 5×7=35 are natural numbers. There is nothing in natural language that is even remotely closer to this mathematical property when we look at the relevant linguistic expressions. Therefore, the following expression in (9), which results from the Merging of 'Amy trusts a man' with 'Amy trusts a man who is known to have three mansions' is ungrammatical:

(9) * Amy trusts a man Amy trusts a man who is known to have three mansions.

Finally, and most importantly, it may also be supposed that the problem of non-termination is in general true of procedures specified in abstraction, given that all procedures in practical reality must terminate, and if so, the problem of non-termination cannot be characterized as a problem for the computational mechanism of grammar. As we shall soon see, this may not be a problem for mathematical functions, or even for the Turing machine, since they are intrinsically mathematical, or purely abstract objects, not anchored in any physical system, though they can be implemented or instantiated in a physical system. But, this does not hold true for the computational mechanism of the language faculty since the language faculty is by its *intrinsic* character a mental system or a mental organization. In fact, the *halting problem* (Turing 1936) that is intrinsic to the model of computation inherent in the specification of the Turing machine must also apply to the putative computational system of the language faculty if the mapping

function of the putative computational system of the language faculty is translated into the operations implicit in the specification of the Turing machine. Even if there could be *intensional* differences between the model of computation implicit in the specification of the Turing machine and the mapping function in standard mathematical formalisms of computability despite the fact that they are descriptively or extensionally equivalent (see Soare 1996), such intensional differences—whatever they turn out to be—cannot be brought forward in order to dodge the halting problem for Turing machines. The reason is that the extensional equivalence between the model of computation implicit in the specification of the Turing machine and the mapping function in formalisms of computability is all that matters to the extrapolation of the halting problem to the putative computational system of the language faculty. Any intensional differences arise from a certain way in which computations are looked at or viewed by humans, and this cannot be built into the language faculty itself. Nor can these differences ground a different *mode* of computational operations that avoids the halting problem, because the problem of non-termination inherent in the halting problem is a fundamental part of any formulation of computation abstracting away from the real world.

One way of demonstrating the problem is to take lessons from the *halting problem* (Turing 1936) that is intrinsic to the model of computation inherent in the specification of the Turing machine. If the language faculty is a computing system, the putative computational system of the language faculty must also face the vagaries of the halting problem. The language faculty in the Minimalist model of Generative Grammar (Chomsky 1995) selects lexical items from the lexicon and then applies the binary operation Merge that combines these lexical items, and finally maps the constructed objects to the sound system (Phonological Form) and the meaning system (Logical Form). It is easy to impose the halting problem on the selection of lexical items from the lexicon in such a manner that the putative computational system of the language faculty may or may not terminate over the selection of lexical items. By following Partee, Ter Meulen, and Wall (1990), we can define the halting problem for the language faculty the following way.

L={*x*: a TM accepts *x*}

Here, L is a language that is defined as a set of *x*s, which are the strings generated/accepted by a Turing machine TM. Let's assume that the *x*s here are discrete lexical items that can be drawn from the lexicon. When FoL is said to terminate on *x*, it actually completes the task of selection of *x* from the lexicon. So, we can have N={*x*: FoL selects and terminates over *x*}, where N is the set of *x*s, and this set can otherwise be conceived of as a list. The halting problem for FoL is simply this: there is no FoL way of telling if

FoL will ever terminate on *x* or not. Suppose that one insists that it is possible. We then have N′={E(FoL): FoL selects and terminates over E(FoL)} where E(FoL) is the encoding of an FoL that is the input to the same FoL itself. Since E(FoL) itself can be in the form of lexical items, just as *x* is, it is okay to have E(FoL) as an input. Since FoL is a computing system, N′ is decidable by FoL. From this, it follows that the complement of N′, which is, say, N′′, is also decidable by some version of FoL, say, FoL′. If so, we have N′′={x′: x′ not the encoding of any FoL, or else x′ is the encoding of an FoL that does not select and terminate over E(FoL). Now we can ask if E(FoL′) is a member of N′′. That is, is E(FoL′)∈ N′′.

If we assume E(FoL′)∉ N′′, then E(FoL′) is not one of the items selected by FoL′, and hence FoL′ does not select E(FoL′). Therefore, FoL′ is a version of FoL that does not accept its own encoding. This ends up making E(FoL′)∈ N′′ true. Contradiction! If, on the other hand, we assume that E(FoL′)∈ N′′, then FoL′ selects and terminates over E(FoL′). But, since E(FoL′)∈ N′′, by our assumption, FoL′ cannot select and terminate over E(FoL′). Hence, FoL′ does not select and terminate over E(FoL′). Again, a contradiction!

The halting problem for the FoL can prove to be fatal because the FoL may never execute computations, as computations get off the ground only when lexical items are combined through Merge. Additionally, this damages the deterministic character of the FoL on the grounds that the computational system that the FoL is supposed to be always maps the outputs of Merge to the sound and meaning systems without fail. The question of the FoL having options in its trajectories or paths of operations does not even arise, also because the computational system operates beyond space and time. Simply speaking, the system of grammar, as a computing device, does not cease to always map syntactic objects to semantic and phonological representations. If this is how computations are always supposed to operate over the symbols that syntactic objects are, then the FoL inherits a deterministic character from the way the linguistic machine functions. But, the halting problem imposed over the selection of lexical items, which actually kick-starts the computational processes in the FoL, undermines this determinism, or rather, this deterministic mode of operation of the FoL. This is guaranteed by the fact that now the FoL may sometimes execute computations or may not terminate on the already started computational processes. This is *not* how the FoL is thought to function since the FoL is assumed to offer an optimal solution to the demands posed by the interface systems (the conceptual and articulatory-perceptual interfaces of the brain/mind) connected to the meaning and sound systems. That is, if the FoL is supposed to ensure a kind of optimization between the computational processes of the syntactic engine and the demands placed on it by the interface systems of the mind, the halting

problem for the FoL eliminates the possibility of the FoL offering an optimal solution to the demands posed by the interface systems. If there is no lexical selection, then there is no computation that can run over the symbols picked up from the lexical inventory. If this happens in some non-deterministic way, then the FoL *may* or *may not* meet the requirements placed by the interface systems, thereby turning into an inelegant system, though it is supposed to be an elegant system of optimization.

Implications

The upshot of the whole discussion in this paper is that grammar does not compute. If grammar does not compute, then there is no reason to think that this conclusion will block humans from producing and comprehending an unbounded number of linguistic expressions. The question of whether and how humans produce and comprehend an unlimited number of linguistic expressions has nothing whatsoever to do with the computational character of the device itself. Let's take the following consideration. While languages may or may not be infinite, in the appropriate mathematical sense of the term, this has nothing to do with whether humans can or cannot produce and comprehend an unlimited number of linguistic expressions (see Lobina 2017). Here, the symbolic nature and form of languages is dissociated from the psychological capacity of humans to make use of the symbolic system of language to produce/comprehend an unbounded number of linguistic expressions. In a similar vein, if grammar, as part of the FoL, which is a mental system, does not compute, then this does not in any way block the possibility that humans can produce and comprehend an unlimited number of linguistic expressions. The reason for this is that grammar, when conceived of as a computing device within the Minimalist model of the language faculty, is taken to be frozen from real space and time constraints, this makes grammar more like a purely abstract symbolic system devoid of contact with the real world. But, the fact that humans produce and comprehend an unlimited number of linguistic expressions has some connection to the real space and time considerations, as humans do process linguistic expressions of an unbounded number *in real time and space*. Therefore, even if grammar as a purely abstract system does not compute, this, in itself, has nothing to allow or disallow the human psychological capacity to produce and comprehend an unlimited number of linguistic expressions. This licenses the conclusion that the psychological capacity of humans to produce and comprehend an unlimited number of linguistic expressions *must* be ultimately segregated from the purely abstract properties of grammar, whether admitting of the conception of grammar as computing device in itself.

This has significant consequences and implications for cognitive computations within language, and other cognitive domains, such as vision, motor systems, etc. Any mental system or domain that is supposed to be language-like, or to possess properties and features (such as systematicity, compositionality, etc.) that are usually attributed to language cannot be said to be computing. Nor can it be regarded as a computing device of some sort. The cognitive system(s) responsible for thought and reasoning can be a good example here, since the system of thought has often been modeled on the system of grammar, as in the well-known Language of Thought (LoT) Hypothesis (Fodor 1975, 2001). Other cognitive domains, such as vision, motor systems, or memory cannot also be regarded as systems that run computations on the grounds that problems orthogonal to the halting problem imposed on lexical selection within the FoL can be extended to these systems. Thus, for example, the visual system can be said to run computations on the selection of properties or features such as size, color, shape, texture, depth, etc. of the perceptual world whose inputs are processed inside the visual system. Likewise, the motor system can be thought to operate on analog values of coordinates of body parts to run the appropriate computations for motor actions, and the memory system can be said to work on snapshots of events and things in order to assemble, de-assemble, sequence, retrieve, and erase memories. In all such cases, the halting problem imposed on the initial process of computations can scupper the computing device a cognitive domain, such as vision or the motor system, is imagined to be. This is more so because cognitive domains such as vision or memory often need to offer sub-optimal solutions to the problems posed by the messy world, and the mappings between designated inputs and outputs are not always straightforwardly driven by content-less properties of the symbols over which computations are supposed to run. That is why various effects, such as visual hallucinations and illusions, false memories, and memory blocking, etc. exist.

There are parallels between this work and the profoundly significant demonstration by the mathematical logician Kurt Gödel that mathematics as a formal system cannot be reduced to any delimited set of principles. Just as mathematics will always remain a never-depleting stream producing new theorems that cannot be bound within any predefined confinements of axioms, the formal system of grammar will always remain an ever-productive system generating newer and newer axioms and constraints of language which cannot be restricted by, and thus reduced to, any limited notion of mental computation.

References

Chomsky, Noam. 1980. *Rules and Representations*. New York: Columbia University Press

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, Mass.: MIT Press.

Chomsky, Noam. 2000. *New Horizons in the Study of Language and Mind*. Cambridge, Mass.: MIT Press.

Fodor, Jerry. 1975. *The Language of Thought*. Cambridge, Mass.: Harvard University Press.

Fodor, Jerry. 2001. *The Mind does not Work that Way*. Cambridge, Mass.: MIT Press.

Foster, Carol Lynn. 1992. *Algorithms, Abstraction and Implementation: Levels of Detail in Cognitive Science*. San Diego: Academic Press.

Lobina, David. 2017. *Recursion: A Computational Investigation into the Representation and Processing of Language*. New York: Oxford University Press.

Mondal, Prakash. 2014. *Language, Mind and Computation*. London: Palgrave Macmillan.

O'Brien, Gerard, and Jon Opie. 2011. "Representation in Analog Computation". In *Knowledge and Representation*, edited by Albert Newen, Andreas Bartels and Eva-Maria Jung, 109-128. Stanford: CSLI Publications.

Partee, Barbara, Alice Geraldine Baltina Ter Meulen, and Robert Wall. 1990. *Mathematical Methods in Linguistics*. Heidelberg: Springer.

Piccinini, Gualtiero, and Andrea Scarantino. 2011. "Information processing, Computation and Cognition". *Journal of Biological Physics* 37:1-38.

Soare, Robert. 1996. "Computability and Recursion". *The Bulletin of Symbolic Logic* 2 (3): 284-321.

Turing, Alan. 1936. "On Computable Numbers with an Application to the Entscheidungs Problem". *Proceedings of the London Mathematical Society* 42 (2): 230-265.