When linguists talk mathematical logic

David J. Lobina

Abstract

As is well-known, Noam Chomsky was greatly influenced by mathematical logic in his construction of a generative grammar for language. Amongst the techniques he borrowed from logicians, recursion has been the most prominent, its role within the theory of language having produced an animated debate in cognitive science in recent years. Despite the fact that most of that debate has not been properly informed by the formal sciences, a recent paper by a number of linguists attempts to clarify what recursion is by anchoring its discussion in the postulates of mathematical logic. Unfortunately, their treatment of all things mathematical contains myriad misunderstandings, misinterpretations, and misrepresentations of the relevant literature. In order to remedy that, I here offer a) a corrective firmly and securely based on mathematical logic; b) an expansion of things recursive as they are conceptualised in computer science; and c) a few historiographical remarks based on (a) and (b) as to why Chomsky may have introduced recursive techniques into linguistic theory.

Keywords: Recursion; Mathematical Logic; Definition by Induction; Mathematical Induction; Computer Science; Recursive Structures.

1 Introduction

As has been chronicled elsewhere (e.g., Tomalin 2006), the advent of generative grammar owes much to mathematical logic, a field that greatly influenced Noam Chomsky in the 1940s and 50s. Amongst the techniques Chomsky borrowed from logicians, recursion is certainly the most prominent, its role within the wider conceptual framework he has espoused in the last 50 or so years having provoked great debate and controversy in recent years.

Prima facie, then, there should be much to commend in Watumull, Hauser, Roberts and Hornstein (2014), given their attempt at clarifying the place of recursion within syntactic theory; I have trudged this very terrain myself, using some of the same sources, and in order to make similar points (e.g., in Lobina 2011, but especially in my doctoral thesis: Lobina 2012b). However, there are so many problematic issues with their paper that extensive commentary is necessary. I will limit myself to two general comments, even though many more could in fact be offered:

1. The characterisation of recursion these authors offer is wholly mistaken, the inevitable result of misunderstanding, misrepresenting, and misinterpreting the relevant literature;

2. Properly understanding the spirit of the times in which Chomsky found himself in the 1940s and 50s can be quite illustrative of why he introduced recursion into linguistics in the first place —and of why he keeps regarding it as such a central property.

In addition, an elaboration of these two points will allow me to bring attention to a number of issues that are either misanalysed by Watumull et al. or simply ignored. Overall, in any case, the essay aims to provide a narrative of the points of contact between mathematical logic and linguistics vis-à-vis recursion, and a discussion of Watumull et al. (2014) offers the perfect opportunity to do so.[1]

2 Recursion and the Formal Sciences

Watumull et al. (WEA, from now on) base their definition of recursion on a quote of Gödel's regarding what they call the 'primitive notion of recursion' (p. 2, cited therein):

> [a] number theoretic function ϕ is said to be *recursive* if there is a finite sequence of number-theoretic functions $\phi_1, \phi_2, \ldots, \phi_n$ that ends with ϕ and has the property that every function ϕ_n of the sequence is recursively defined in terms of [...] preceding functions, or [...] is the successor function $x+1$.

From this citation, WEA take the concept "recursion" to encompass three properties: a) a recursive function must specify a finite sequence (Turing computability, WEA claim); b) this function is recursive if it is defined in terms of preceding functions, that is, if it is defined by induction, which WEA associate with strong generativity (i.e., the generation of ever more complex structure); and c) this function may in fact just reduce to the successor function (that is, mathematical induction, which WEA associate with the unboundedness of a generative procedure). Unfortunately, this characterisation of recursion is mistaken in both design and detail.

To begin with, Gödel is therein not defining recursion per se, but a specific class of functions —they are two different things, as I will show below. As Davis (1965, p. 4) points out in his introduction to the 1931 paper WEA focus on,[2] what Gödel calls therein *recursive functions* are now known as the *primitive recursive class*, a revised terminology introduced by Kleene in 1936 (also reprinted in Davis 1965). This is an important point, as WEA begin this section of their paper by pointing to

[1]Both Tomalin (2007) and Fitch (2010) discuss the influence of mathematical logic on linguistics, but my take on things differs from theirs in some respects. In particular, the latter identifies recursion with a self-embedding operation (putting things inside other things), and I consider this a clear mistake, while the former argues that linguists should stop using the term recursion and use the expression "inductive definition" instead, but for unpersuasive reasons, in my opinion (see Lobina 2014 for discussion of these two papers).

[2]Gödel's article is reprinted in Davis (1965), an anthology of foundational papers in mathematical logic and the source I will be using here.

the 'mathematical equivalence' between computability and Gödel's recursiveness (1934, cited therein), this statement then followed by the definition we have cited from the 1931 text. This is rather misleading, however. In his 1934 paper, Gödel expanded the primitive class of recursive functions of his 1931 paper into what is now known as the *general recursive functions*, and it is only the latter, and certainly not the former, that have been claimed to formalise the class of computable functions (see Epstein and Carnielli 2008, p. 104 for a succinct demonstration of this claim). What's more, we not only know that the primitive class is not computable, we also now know that the general class of recursive functions cannot be regarded as a correct formalisation of the computable functions either (see infra).

In any case, and more importantly, if we provide the full version of the 1931 quote, it is demonstrably clear that Gödel is outlining which functions are (primitive) recursive; he is certainly not listing the 'three criterial properties of recursion' (WEA, p. 1; I'll be quoting from Davis (1965), which offers a different translation from the one WEA use, but this won't affect my analysis).

> A number theoretic function ϕ is said to be recursive if there exists a finite sequence of number-theoretic functions $\phi_1, \phi_2, \ldots, \phi_n$ which ends with ϕ and has the property that each function ϕ_k of the sequence either is defined recursively from two of the preceding functions, or results [footnote not included] from one of the preceding functions by substitution, or, finally, is a constant or the successor function $x+1$ (pp. 14–5; underline in the original).

As is clear from the full quote, WEA are leaving out a significant amount of detail, which might well make one wonder whether the criterial properties they identify are more than just three —whatever happened to substitution or being a constant? Nevertheless, it is clear that nowhere in this quote is Gödel actually offering any such criterial definition. Here Gödel is merely saying that if we have a list of functions (and there is no indication that this list is computed by a Turing Machine, a notion that was unavailable to Gödel in 1931 anyhow), any one function from this list will be defined as recursive if it

- is defined by induction from previous functions, OR
- is substituted by some of them, OR
- is the constant function, OR
- is the successor function.

Clearly, Gödel's definition is not a combination of properties subsuming a specific concept; rather, it's an *either. . . or* type of definition listing the various primitive recursive functions. In fact, Gödel's quote became a pretty standard definition of the primitive recursive class, as any textbook from that time can show. Kleene (1952), for instance, outlines five primitive recursive functions (or schemata, as he calls them; pp. 220–3) and the close correspondence between these five schemata and Gödel's definition ought to be apparent: schema 1 is the *successor function*, schema 2 are the *constant functions*, schema 3 are the *identity functions*, schema 4 Kleene terms a *definition by substitution*, and schema 5 is *primitive recursion*.

As mentioned, Gödel expanded the primitive recursive functions into a more general class of recursive functions in 1934. At the time, Church was developing his λ-calculus in order to investigate the foundations of logic (Church, 1932), but in a letter dated 1934, Church suggested to Gödel that computability should be identified with the general recursive formalism. Though this proposal was not well-received (Sieg, 2006, p. 191), in a 1936 paper, also referenced by WEA, Church nevertheless turns his attention to the general recursive functions proper, identifies them with the class of computable functions, and shows their equivalence to λ-definability. WEA mention this and point out that Turing computability was also shown to be equivalent to these other formalisms. This is factually true, as far as scholars knew in the 1930s at least, but it is also somewhat misleading given that the formalisms based on the λ-calculus and the general recursive functions were at the time in fact inadequate for the purposes at hand, as we now know.

Indeed, the λ-calculus was soon discovered to be inconsistent (i.e., it gave rise to paradoxes) and it was only fixed for the purposes Church had in mind much later; in particular, if the part that deals with logic (such as logical constants, etc.) is eliminated, then this results in an "untyped", pure λ-calculus that is relevant to the formalisation of a computation (Barendregt, 1997). As for the general recursive functions, the fatal flaw seems to be that the core of Church's characterisation consisted in a number of atomic steps that were stepwise recursive,[3] making it semi-circular (Sieg, 2006, p. 193), and he provided no justification for this (see Soare 1996, p. 289–91 and Sieg 1997 for details). It was in fact Church's student, Kleene, who replaced the general recursive with the *partial recursive functions* in a 1938 paper, and this class correctly formalises the computable functions.[4]

WEA are a bit careless in their discussion, for they present the results of the 1930s as if they were still valid today, and this is not entirely correct. I am not suggesting that Church et al. failed to prove the equivalence of various formalisms;[5] what I am pointing out is that some of those formalisms failed to capture the class of computable functions, and that is a crucial failing. For present purposes, the formalisms that correctly formalise computability and that furthermore converge on the same input-output relations (making them extensionally equivalent to each

[3]It is important to note that it is Church's characterisation of the general recursive functions that is relevant here, not Gödel's; the latter never tried to prove that the general class of recursive functions was computable, and it is not clear he ever accepted any of Church's claims (see Sieg 2006 for discussion).

[4]See Epstein and Carnielli (2008, p. 126) for some comments regarding how the general and the partial classes relate, including the limitations of the general class when it comes to formalising the computable class of functions.

[5]Rosser (1939) appears to have been the first to have shown the equivalence between general recursion, λ-definability, and a Turing Machine, but see the discussion in Sieg (1997) for doubts regarding the correctness of that result.

other) are the partial recursive functions, an untyped λ-calculus, and Turing computability (others can be added, but they don't matter now). I will come back to the comparison of these formalisms below.

Thus far, then, we have three different classes of recursive functions: the primitive, the general, and the partial. It is important to note that all these functions are recursive for the very same reason (see immediately below); they are different classes of functions because of the different input-output relations they encompass. Thus, it would be a mistake to conclude from this that recursion as a concept is different for each class of functions. Admittedly, one could claim that the *form* of recursion subsuming each class is different, but no more than that should be granted. In any case, WEA are certainly mistaken to describe Gödel's definition of the primitive class of recursive functions as a primitive notion of recursion. After all, just as the general or partial classes don't provide a general or partial notion of recursion (surely no-one would want to claim *that*), a definition of the primitive recursive functions doesn't itself yield a primitive notion of recursion.

So what is recursion, then? As Brainerd and Landweber (1974) put it, it is useful to define functions 'using some form of induction scheme..., a general scheme... which we call recursion' (p. 54). This is in fact consonant with the original interpretation of recursion as being part of a definition by induction, as chronicled by Soare (1996). Also known as a recursive definition, it consists, as WEA themselves allude to (e.g., in page 2), in 'defining a function by specifying each of its values in terms of previously defined values' (Cutland, 1980, p. 32). WEA don't actually provide an example of what a definition by induction involves, and it is important to do so. A standard example is that of the factorial class ($n! = n \times n-1 \times n-2 \ldots \times 2 \times 1$, where n is a natural number), which can be recursively defined in the two-part system so typical of such definitions: a) if $n = 1$, $n! = 1$ (base case), b) if $n > 1$, $n! = n \times (n-1)!$ (recursive step). Note that the recursive step involves another invocation of the factorial function. Thus, in order to calculate the factorial of, say, 4 (i.e., $4 \times 3!$), the function must return the result of the factorial of 3, and so on until it reaches the factorial of 1, the base case, effectively terminating the recursion. Thus, self-reference is what makes a function recursive (Tomalin, 2006, p. 61), it is its defining feature —its denotation, if you will, making it a *special type* of recurrence.

This much is implicitly present in WEA towards the end of their paper, when they talk of a set being 'defined recursively (i.e., by induction)' (p. 6), which makes for a curious reading experience, considering that a definition by induction is supposed to be a criterial property of recursion itself, so how comes it that recursion is described as being part of what a definition by induction is? As it happens, recursion is also part of the third property WEA highlight: mathematical induction (sometimes called an inductive proof or an inductive definition), a similar but distinct concept to a definition by induction. A mathematical induction is a technique employed to prove whether a given property applies to an infinite set, and proceeds as follows: first, it is shown that a given statement is true for 1; then, it is assumed that it is true for n, a fixed number (the inductive hypothesis); and finally, it is established that it is therefore true for $n+1$ (the inductive step). It is important to note that inductive definitions are a central feature of recursive definitions in the sense that the former grounds the latter; that is, mathematical induction justifies the recursive definition of a function over the domain established by the inductive definition (Kleene, 1952, p. 260). Thus, whilst being similar —both make use of recursion, after all— a definition by induction and mathematical induction are different concepts.[6]

After these brief comments, we are now in a position to pinpoint what's wrong with WEA's characterisation of recursion. I have already alluded to the first property WEA list, but the criticism is worth expanding a little bit: there's no reason to equate the finite sequence of functions Gödel mentions in his 1931 paper with Turing computability; nor is there any indication in that text that such a sequence of functions is the result of a computation (and in any case, that is not an identity condition for the concept of recursion). As for the second property WEA outline —the identification between a definition by induction and structure generation— these are clearly two very different things. It is not clear why WEA make the connection at all, actually, but perhaps there is a hint of what is going on in the manner in which they describe what a Turing Machine (TM) does, a description that is not standard (to say the least).

Indeed, WEA talk of the outputs of a TM as being 'recursed' rather than returned, the chain of such operations involving 'increasingly complex structures [being] carried forward on the [TM] tape' (p. 2).[7] As shown above, though, a function is recursive if it's defined in terms of previously defined values, and this is unrelated and unrelatable to the manner in which WEA describe what a TM does. This is clearly the case for the factorial class, or indeed for any other recursively-defined functions; previously defined values are not objects that partake in the further construction of other, more complex objects. Rather, the computation of the factorial of 4, for instance, necessitates the computation of the factorial of 3, but

[6]Note, in particular, that a mathematical induction contains an inductive hypothesis, whereas a recursive definition doesn't.

[7]On this same page, WEA quote from Chomsky (1975, p. 67) (mistakenly listed as a book published by Plenum in 1955 in the bibliography; it was certainly written in 1955-6, but Plenum published an *edited* version of it in 1975) that a derivation can be regarded as a "running through" of rules, adding within brackets that the word *recursion* derives from the Latin *recursio* (actually, it derives from *recursus*), the latter meaning 'a running back' (actually, WEA say it means "running back", but not even that is correct, given that *recursio* is a noun, and hence can only mean "a running back", as defined in *Andrew's Edition of Freund's Latin dictionary*). Considering that Chomsky (1975, p. 67) is merely saying that rewriting rules (using the format of a Post production system, as in Post 1944) can be provided for the construction of syntactic structures, and that a derivation is a running through of such rules, it must be wondered why they bother to connect that to the meaning of the original word in Latin. The connection between the two is as random as one can imagine, and it carries no weight whatsoever —unfortunately too often an occurrence in their paper. In any case, I noted the etymological origins of the English word *recursion* in Lobina and García-Albea (2009), and pointed out that even though it was used in the past with the same meaning as its Latin precedent, that connotation was obsolete by the 1930s. It was of course precisely at that time that mathematicians were starting to use it with a rather different connotation.

the latter is the value that is calculated by another function, neither internal to the factorial of 4 nor constitutive of its operations.

This is also clear in the case of a TM, actually; all a TM does, after all, is write or erase digits on a tape according to some rules (the so-called configuration), and whilst a collection of digits could stand for many different things (as specified in the TM's look-up table at least), every operation a TM carries out is exhausted at each stage. That is, the TM's configuration specifies what its control structure (the read/write component) does when scanning a given cell in the TM tape, but the configuration changes for each cell, making each step of a TM computation a self-contained one. As such, a TM is an iterator, as is usually termed in careful characterisations of computational mechanisms (for instance, in Moschovakis and Paschalis 2008). Therefore, there is neither structure nor 'derivational history' (WEA, p. 4, ft. 2) being carried forward on its tape; to describe the operations of a TM in such terms is to talk in metaphors.

Moreover, it is rather inaccurate to describe the operations of a TM as "recurses", as this suggests recursion plays a role in the computations a TM carries out, and this is not the case.[8] The extensional equivalence between a TM and partial recursive functions was mentioned above, but this simply refers to the fact that they can compute the same input-output relations, it does not pertain to the manner in which these input-output pairs are in fact effected/computed. As a matter of fact, each formalism computes input-output pairs in a different way, these being the intensional differences among them, as discussed in Soare (1996).

A similar state of affairs applies to the three-way identification WEA specify regarding the third criterial property of recursion: the successor function qua mathematical induction qua the unboundedness of generative procedure. The successor function, as one can appreciate from the Gödel quote that started our discussion, is a (primitive) recursive function and therefore it is related to mathematical induction in the same way that other recursive definitions are. However, the misidentification between mathematical induction and the successor function could perhaps be excused on the grounds that most expositions of mathematical induction employ the successor function as a case in point (as in Kleene 1952, pp. 20 et seq., for example). This doesn't have to be the case, though; Buck (1963) provides examples of mathematical induction with other types of data. In any case, what mathematical induction clearly isn't is unboundedness itself; as stated above, mathematical induction is a technique to prove if a given statement is true of an infinite set, a rather different concept altogether. I don't, of course, mean to claim that the concepts of the three-way identification WEA posit are entirely unrelated; after all, an implementation of the successor function in a computer programme could well result in an unbounded process that is infinite in character. My point, rather, is that these

[8]The mistaken use of this vocabulary allows WEA to employ the term *tail recursion* for when a TM only operates over the 'end. . . of the immediately preceding output' (WEA, p. 4, ft. 2; it's not clear what that even means, though), but this is unwarranted, for the reasons just espoused. In any case, the term *tail recursion* is widely employed in computer science to refer to something rather different, and I have never encountered the usage WEA employ in the mathematical logic literature.

three constructs are not one single phenomenon, and it is a mistake to treat them as such.

Let me recap and restate this part of the paper, for it is important to be as clear as possible. WEA's definition of recursion is triply mistaken, and at different levels to boot:

1. They misinterpret a text of Gödel's as providing a definition of 'the primitive notion of recursion', when in fact Gödel had defined the class of primitive recursive functions;[9] what's more, they misrepresent this quote so that only three properties are to be extracted, when the actual text says more than that —and eventually something else altogether.

2. WEA clearly read too much into the quote, misidentifying a finite sequence of functions with Turing computability (the first criterial property of recursion) and mistakenly equating the successor function with mathematical induction (the third).

3. Two unwarranted connections are also put forward: a) between a definition by induction and strong generativity (the second property) and b) between mathematical induction and the unboundedness of a generative procedure.

All in all, therefore, WEA's attempt at conceptual clarity is a disaster, its demonstrable incompetence in matters mathematical only matched by the self-assurance of its claims. In particular, the way in which they use Gödel's quote is very worrying indeed; it is not only bad scholarship, it borders on wilful misrepresentation. Indeed, why choose a definition of the primitive recursive functions, in any case? Why not use a definition of the general class instead? Or the partial class? And why focus on that particular text of Gödel's? It may all appear a bit random, to be sure, until one realises that the three properties WEA want to draw our attention to can be easily extracted from Gödel's quote, but in a selective way only.[10]

I should note, in any case, that I don't dispute the importance of the three properties WEA line out for a theory of language. I certainly think that a generative procedure that builds ever more complex structures with no arbitrary limit on its operations is a necessity.[11] What I deny is what they actually claim (WEA, p. 2): that these three properties are criterial of what recursion is, and more specifically, that these properties are associated to (or indeed identified with) the three

[9]In any case, Gödel hadn't called these functions primitive recursive, nor was the term in use in 1931.

[10]In my opinion, that such a piece could ever be published reflects badly on both the journal that accepted it and the referees that reviewed it. Or perhaps it reflects the nature of this journal just fine; according to their website, WAE's article could have come with a publishing fee of 1600 euros (`http://www.frontiersin.org/Language_Sciences/fees`).

[11]Although proving that the language system generates an infinite set of sentences is not a trivial matter; see Langendoen (2007, 2010) and Pullum and Scholz (2010) for relevant discussion —the latter, in fact, discuss whether one could prove the infinity of language via mathematical induction, but find the strategy wanting.

constructs WEA selectively extract from the aforementioned and discussed quote from Gödel (1931).

Be that as it may, the issue of the intensional differences among the formalisms I have mentioned deserves some attention, as its discussion will provide us with useful information regarding the historiographical remarks I would like to end this paper with. At one point, WEA state that Turing 'formalized the intuitive concept of a computation —equivalently, a *generative procedure*' (p. 1), a statement that is then followed by the aforementioned claim that various formalisms are in fact equivalent. Given the manner in which WEA set up the discussion, one can sort of understand the general gist of their viewpoint: a correct specification of a generative procedure, or algorithm, has been obtained, and this was (and still is) of great relevance to linguistics. Such a take on things has certainly produced colourful language in the past. Kleene (1952, p. 320) thought that the equivalence amongst these systems suggested that we are dealing with a fundamental class, whilst Post (1936, p. 291) saw it as a "natural law". Perhaps no more sober-mindedly, Epstein and Carnielli (2008) call the extensional equivalence the The Most Amazing Fact, given that '[a]ll the attempts at formalizing the intuitive notion of computable function yield exactly the same class of functions' (p. 85). As ever, things are not as straightforward as they seem.

The aforementioned studies from the 1930s and 40s did not, contrary to popular belief, settle what a computation is. In the case of the models proposed by Church and Kleene, their work should properly be seen as an attempt to clarify the nature of the functions that can be computed —that is, it is a hypothesis concerning the set of computable functions— but there is more to an algorithm than the function it can compute (Blass & Gurevich, 2003). Turing's idea of a computing machine, on the other hand, does not model an algorithm either; what this formalism describes is what actually happens during a computation that is set in motion, it is an example of a model of computation, as Moschovakis and Paschalis (2008) put it.[12] As such, then, that these formalisms haven't really captured what an algorithm qua abstract mathematical object actually is (as opposed to formalising a procedure or a computational process), an endeavour that is of some foundational significance (see Dean 2007 for discussion of what this author therein calls algorithmic realism).

What lies at the heart of this point is the fact that the intensional differences among the various formalisms involve rather different intuitions as to what a computation is. This has been hinted at in various places, in fact; Soare (2009), for instance, notices that 'Post's (normal) canonical system is a generational system, rather than a computational system. . . because it gives an algorithm for generating (listing) a set of integers rather than computing a function' (p. 380), whilst Church himself, in a review of Turing's work (reprinted in Sieg 1997), points out that even though Turing's formalism has the advantage of making the identification between a computation and a TM intuitively obvious, the other solutions (that is, general

[12]A model of a computation is also sometimes called the implementation of an algorithm, or a computation in course.

recursiveness and the λ-calculus) are more suitable 'for embodiment in a system of symbolic logic' (p. 170). As far as I know, however, Moschovakis (2001) has been the most explicit on this issue, defending the idea that an algorithm is better described in terms of the recursive equations he inherits from McCarthy (1963), while machines (such as a TM) model implementations instead (see Moschovakis 1998 and Moschovakis and Paschalis 2008 for more details).

It seems to me that these intensional differences may carry a rather substantive implication for cognitive science. Elsewhere (Lobina, 2012a), in fact, I have argued that it is not surprising that recursively-specified algorithms have featured so extensively in linguistics (as in the work of Chomsky) while a TM has been widely adopted by much of cognitive psychology (as in Fodor 1983, for instance), for the sort of computations the former field has outlined are certainly more abstract than the real-time computations the latter discipline claims are at play in behaviour. That is, a model of a computation appears to be a more fitting format for describing the online computations of processing systems, linguistic derivations requiring something else, considering that a TM does *not* generate structure in the sense that linguists understand this notion. The differing computational intuitions the various formalisms give rise to also point to differences between mathematical logic and computer science, and a few comments are in order at this point, given the direction my discussion is taking.[13]

Perhaps an obvious distinction between mathematics and computer science, simplifying somewhat, is that the former deals with mapping functions, while the latter focuses on algorithmic implementations, programs and procedures, and this distinction involves different theoretical motivations. In the case of computer science, implementations and procedures must meet a number of conditions, such as Mal'cev's well-known five criteria: discreteness, determinacy, elementarity of the steps, direction and massivity (see Epstein and Carnielli 2008 for further details; cf. the five criteria in AOCP, pp. 4 et seq.).[14] As Roberts (2006, p. 47) states, this is not the case for functions, for they are not algorithmic in nature, while procedures are —functions need not be effective, after all (SICP, p. 21).

Consider the following, extremely schematic description of how a computer scientist interacts with a computer language via what is usually called an *interpreter*. We type an *expression*, either simply a primitive element or two or more elements linked by an operator, and the interpreter *evaluates* the expression according to the rules specified in a given procedure.[15] As such, procedures can apply

[13]I will be relying on two sources here: Abelson and Sussman 1996, SICP henceforth, and Knuth 1997, vol. 1, AOCP hereafter.

[14]A program is here understood as a 'machine-compatible representation of an algorithm' (Brookshear, 2000, p. 2); or as Knuth puts it: 'the expression of a computational method. . . is called a program' (AOCP, p. 5). A *procedure*, on the other hand, will be treated here as a step-by-step list of instructions for completing a task, a definition that is sometimes informally applied to algorithms. The literature sometimes employs these two terms —algorithms and procedures— interchangeably, but I follow SICP and AOCP in distinguishing them.

[15]I won't discuss the various evaluation strategies a programmer may employ; these don't matter here.

recursively. That is, in order to evaluate the elements of a compound, the interpreter must evaluate all the elements of the sub-expressions, which involves applying the procedure of the operator (i.e., the leftmost element of the sub-expression, following Polish notation) to the operands (i.e., the rightmost element(s) of the sub-expression). In order to evaluate a complex expression, then, the interpreter must evaluate each element of the sub-expression first. Thus, the evaluation rule contains as one of its steps the invocation of the rule itself.

Of perhaps more interest is the nature of the processes that procedures generate, a difference between the actual computation in motion and the meta-rules that direct the process. According to SICP (pp. 33–4), recurrent processes (or implementations) can be either recursive or iterative. Recursive implementations involve a self-reference (a given operation calls itself) and as a result chains of unfinished tasks develop. In the case of iteration, an operation is repeated in succession, and in general its state can be summarised at any stage by the variables plus the fixed rule that establishes how these are updated from one state to another. If no termination conditions are established, both processes will proceed indefinitely. Furthermore, whilst both types of processes keep something in memory, recursive processes keep deferred *operations* rather than just *variables*, and this usually exerts a bigger load. Note that it is perfectly possible for a recursive procedure to produce an iterative process (see SICP, pp. 33–4, for an example). There is no contradiction here; it is the actual rules laid out in a procedure rather than how these are defined that establish the nature of the generated process. As a result, there is a certain subtlety involved in computing with a recursive definition —and in distinguishing between procedures and processes.

As it happens, however, all tasks that can be solved recursively can also be solved iteratively; or in other words, all recursive relations can be reduced to iterative relations (Rice, 1965).[16] Be that as it may, recursive implementations are specially well-suited to operate over complex objects such as a "recursive data structure", defined by Rodgers and Black (2004) as an object or class 'that is partially composed of smaller or simpler instances of the same data structure'. That is, a structure that includes an abstraction of itself (an X within an X), and "trees", "lists" and the like constitute the prototypical cases (trees inside other trees, or lists inside lists, etc.). In fact, a natural fit between recursive data structures and recursive mechanisms is stressed in no small measure by Wirth (1986, p. 135). Despite this close correspondence, orbiting conditions —such as memory limits, architectural complexity, efficiency— more often than not bring about iterative implementations. The orbiting conditions traditionally have to do with memory limitations of the physical machine that is implementing the algorithm; i.e., properties of the implementation and not of the algorithm itself. Therefore, it can be the case that even though a set of data structures naturally merits a recursive implementation,

[16]This point already follows from the extensional equivalence of partial recursive functions and a TM, though. That is, any input-output pair that can be computed with a recursive function can also be obtained, iteratively, by employing a TM.

iteration is chosen instead; after all, implementations require time and space and so come with inherent practical limitations in terms of how they can actually be instantiated.

According to computer science, then, recursive structures and recursive mechanisms are independent constructs, their connection a matter of empirical investigation, not conflation. As I have argued elsewhere, the conflation between recursive mechanisms and recursive structures is the main problem with the manner in which cognitive science at large treats the concept of recursion. Indeed, many scholars seem to believe that recursive structures can only be generated by recursive operations, or that recursive operations can only generate recursive structures, but this is certainly not the case (see Lobina 2011, 2014 for ample discussion, examples a-plenty).

This is an important point, and it is entirely missed by WEA, mainly because they don't allow for such a possibility at all. A point that keeps coming up in WEA is that the recursiveness of a function is defined independently of its output (p. 1), and therefore equating recursion to the properties of an output is either a fallacy (p. 3) or a mathematical error (p. 4). Admittedly, mathematical logic doesn't make much use of recursive objects; the closer it gets to defining any object as such are the recursively enumerable and the (general) recursive sets, and even these are so defined in terms of how they are generated and not in their own terms. Indeed, a set is recursively enumerable if there is a mechanical procedure that can list/enumerate all its members, while a set is (general) recursive if there is an algorithm that can determine whether a given element is (or is not) one of its members (Post, 1944). Thus, mathematicians restrict the recursive property to functions or rules, it is never applied to objects.

It seems to me, however, that adopting WEA's stance in the study of language makes for a rather rigid, if not obtuse, position to hold (cf. Collins 2008a, pp. 160–1 for a similar view to that of WEA's). I certainly fail to see why linguists should be constrained by the practices of mathematicians at all; linguistics, after all, is not mathematical logic. Clearly, linguists ought to employ whatever tools from the formal sciences may be deemed to be useful —as long as they are used appropriately, of course. If that means borrowing from computer science, as in the postulation of recursive structures, so be it.

In actual fact, this seems to be the case already, albeit only implicitly so. It has been a long-running point of generative grammar, after all, that the linguistic system is governed by structural properties, these being either mechanisms that operate over particular structures (derivational theories) or constraints that licit specific configurations of syntactic objects (representational theories). Indeed, linguistics is full of studies that rightly focus on structures, in many cases entirely independently of whatever function generates them, and one would as a result be hard pressed to justify why equating recursion to certain structures should not be allowed —at least as long as recursive structures are defined in their own terms. Such definitions have been part of the theory since the beginning, in fact. Chomsky and Miller (1963, p. 290) defined constructions in which a self-embedded phrase appeared on either side of a sentence as either right- or left-recursive structures (with no connection whatsoever to either right- or left-recursive rules, whatever these might be), whilst Chomsky (1965) drew a distinction between nested constructions and self-embedded sentences.[17] In particular, the distinction between recursive mechanisms and recursive structures is a useful one in linguistics, one that has featured in the computer science literature extensively, and it would be rather spurious to claim that there is a mathematical error (let alone a fallacy) in drawing it —certainly one wouldn't want to levy such charge at computer scientists. In any case, WEA's point misses its target completely. They offer such reasoning in order to refute those studies that claim that if a specific language lacks recursive structures, then recursion is not part of the grammar for that language —not part of its generative procedure, that is. That argument *is* fallacious, but not for the reasons WEA state (for example, see Tomalin's 2011 discussion of Everett's work, cited therein); after all, it is not obvious that the scholars WEA mention actually accept the mathematical practice of how functions relate to their outputs, or that they even accept that the generative procedure underlying language should be regarded as a *mathematical* function, and thus they couldn't possibly be held responsible for committing the errors or fallacies WEA accuse them of.

So what is recursion, again? Self-reference, I submit, is the property that underlies what recursion is, it is its denotation. As such, it can apply to different theoretical constructs, its connotations, thereby accounting for the fact that we have recursively-defined functions in mathematics (i.e., functions defined in terms of previously calculated values), and recursive procedures (evaluation procedures that involve architecturally-equivalent sub-operations), recursive processes (wherein operations call themselves, generating chains of deferred operations), and recursive structures (objects in which an X is constitutive of a larger X) in computer science. The real issue is to not conflate these notions, for it is that error that creates the most confusion in the literature.

3 Some historiographical remarks

Many scholars will know that the use of recursion has proved to be problematic in mathematical logic too, as Soare (1996) has so clearly shown. Therein, Soare describes a state of affairs in which recursion is taken to be at the very centre of what a computation is, to the point that systems of recursive equations, or recursion itself, are employed as almost synonymous with computability and/or computable, in detriment of Turing's model. He suggests this is the result of the literature tacitly following what he calls the Recursion Convention (RC): *a*) use the terms of the general recursive formalism to describe results of the subject, even if the proofs are

[17]To wit: 'phrases *A* and *B* form a nested construction if *A* falls totally within *B*, with some nonnull element to its left within *B* and some nonnull element to its right within *B*', whilst 'phrase *A* is self-embedded in *B* if *A* is nested in *B* and, furthermore, *A* is a phrase of the same type as *B*' (Chomsky, 1965, p. 12).

based on the formalism of Turing computability; *b*) use the term Church's Thesis to denote various theses; and *c*) name the subject using the language of recursion (*recursion function theory*).

Soare has certainly amassed much evidence supporting this description of the field, and his point is well-taken (see Soare 2007a, 2007b, 2009 for further elaboration). However, I think he misses the point about the differing computational intuitions the various formalisms give rise to, his emphasis on Turing's formalism as a result being a bit misplaced. As suggested above, a TM is more appropriately a model of a computation, or an implementation, rather than a specification of an algorithm. It was also remarked above that Moschovakis (2001) defines an algorithm in terms of systems of recursive equations —an algorithm is a *recursor*, in his terminology— and while I do not wish to legislate as to which view is actually correct, I would like to point out that the work Moschovakis and his colleagues are conducting does not seem to be another instance of the RC.

Why do I mention the RC, though? As a disclaimer, let me state that I have studied the role of recursion in linguistics for the last seven or so years, and I have always thought that Chomsky's emphasis on this notion was in a way puzzling. I understood well his point about providing a generative account of language, and I could also as a result recognise his looking to mathematical logic in order to achieve that. However, why focus on recursively-specified formalisms rather than on a TM or the lambda-calculus? Indeed, Chomsky's consistency on employing recursive techniques is remarkable indeed.

Ever since his 1975 book (recall, actually written in the 1950s), Chomsky has claimed that it is the 'recursive character' of phrase structure rules that allows for the 'generation of infinitely many sentences' (pp. 171–2). In the early years of generative grammar, Post's production system was employed to generate syntactic structures, a formalism that is underlain by recursion, as Chomsky and Miller (1963, p. 284) explicitly recognised.[18] In Chomsky (1965), it is stated that a grammar 'must meet formal conditions that restrict it to the enumeration of recursive

[18]WEA (p. 3) quote the relevant excerpt from the page I have just cited to make this very point, but this had already been explicitly discussed in Lobina (2011, p. 154) and Pullum (2011, p. 288), among other places. Production systems can be reduced to what Post calls a "normal form", which can be described in terms of the mapping from gP to Pg' (the transformation mediated by an arrow or the word *produces*), where g stands for a finite sequence of letters (the *enunciations* of logic, the actual subject matter of Post) and P represents the operational variables manipulating these enunciations (Post, 1943, p. 199). Crucially, the whole approach 'naturally lends itself to the generating of sets by the method of definition by induction' (ibid., p. 201). The general recursive property of production systems should not be confused with an internal application of recursion within specific rules, as in those cases in which the same symbol appears on both sides of the arrow, such as in the rule $NounPhrase \longrightarrow Noun + NounPhrase$, an application particular to formal language theory and linguistics, but unbeknownst to Post. On another front, Chomsky (1956) showed that production systems could be employed to characterise different collections (classes) of *formal grammars* and the sequences of symbols (also called strings; these sequences constitute the corresponding formal languages) that these grammars are said to generate/produce. A ranking can then be so devised as to classify these grammars in terms of their *expressive power*, but this is not an issue I will be discussing here.

sets' (p. 208), and similar statements are offered in the first edition of his *Language and mind* book, re-edited in 2006 but originally published in 1966: 'in general, a set of rules that recursively define an infinite set of objects may be said to generate this set' (p. 112); and, 'generative grammar recursively enumerates structural description of sentences' (p. 165).[19] The focus on the recursive enumeration of linguistic objects was maintained in Chomsky (1981, pp. 11–3) and re-emphasised in the 1990s with the postulation of a computational mechanism, termed *merge*, that 'recursively constructs *syntactic objects* from [lexical] items...and syntactic objects already formed' (Chomsky, 1995, p. 226). A recent description delineates *merge* in very general terms as a set-theoretic operation in which repeated applications over one element yield a potentially infinite set of structures, drawing an analogy between the way *merge* applies and the successor function; the latter underlies what is known as the "iterative conception of set" (Boolos, 1971), a process in which sets are 'recursively generated at each stage' (ibid., p. 223).[20]

During the writing of Lobina (2011), to keep with the personal reflections of this section, I often wondered what exactly justified the insistence on recursively-specified algorithms, as I couldn't really find any justification in the literature. I eventually recognised that the extensional equivalence of the various formalisms to be found in mathematical logic was of no relevance whatsoever, given that the subject matter of linguistics had been delineated as a study of the *function in intension* that generates sound-meaning pairs.[21] That is, the particular way in which these pairs are generated is all-important, and therefore there is a genuine question as to which formalism is the correct one. However, there is no demonstration anywhere in the literature that a recursively-specified mechanism is to be preferred to a TM or to any other formalism; moreover, the theory has moved from production systems to set-theoretic operations such as *merge* as if the differences between these recursively-defined formalisms didn't matter, and that doesn't seem to be right. Eventually, I started considering the possibility that Chomsky may have been influenced by Soare's Recursion Convention; not at all far-fetched, I thought, considering the spirit of the age Soare had described and Chomsky's own background. I decided to ask Chomsky directly, and given that his response was rather enlightening, I included it as a footnote in my 2011 paper. Both the paper and the footnote have gone almost entirely unnoticed in the literature, however, so perhaps it would be useful to cite part of the footnote here (the conversation took place in May 2009, I'm quoting from two different emails):

[19]Actually, the 1965 quote should read "the recursive enumeration of sets" to be compatible to those extracted from the 1966 book, as an enumeration of recursive sets is something rather different indeed (recall the definitions of recursively enumerable sets and recursive sets supra).

[20]I drew attention to Boolos's paper in Lobina (2011, p. 154), where I used a slightly longer citation; Watumull (2013, p. 309) uses this very quote to make the same point, almost verbatim.

[21]This is explicitly defended in Chomsky (1980, p. 82), and there is a consensus of sorts in philosophy of language as to its correctness (see Collins 2007, 2008b; Matthews 2006; Smith 2006). The distinction between *functions in intension* and *functions in extension* was introduced by Church (1941).

> "[T]here is a technical definition of 'recursion' in terms of Church's thesis (Turing machines, lambda calculus, Post's theory, Kleene's theory, etc.)", the only one used he's ever used, "a formalization of the notion algorithm/mechanical procedure".
>
> "I have always tacitly adopted the Recursion Convention".

I won't discuss here whether any of this makes the place of recursion within generative grammar a less secure one, but perhaps the implications are clear. In any case, that seems to settle Chomsky's aims and motivations regarding the place of recursion in his theory of language.[22]

I would like to end this paper by bringing attention to an aspect of WEA that I find rather self-serving, to say the least. In the abstract, WEA point out that the hypothesis put forward in Hauser, Chomsky and Fitch (2002) regarding the uniqueness of recursion 'was based on the standard mathematical definition of recursion as understood by Gödel and Turing' (p. 1). That statement should be a surprise to anyone who knows the literature well, for that claim doesn't appear in either the original paper itself or in Fitch, Hauser and Chomsky (2005), the latter a response to some criticisms. I don't deny that that was probably Chomsky's take on the issue, but it certainly wasn't for either of the other two authors, one of whom is also an author in WEA. As is clear in Hauser (2009) and Fitch (2010), these authors identify recursion with an embedding, or self-embedding, operation (an operation that puts an object inside another, bigger object of the same kind), and *that* is unrelated and unrelatable to Chomsky's understanding of recursion —or to any aspect of mathematical logic for that matter. In a similar vein, I should also point out that another one of the authors in WEA has elsewhere defined recursion as an operation that applies over its own output (Hornstein, 2009), a trivial property of recursive processes, of course, but not one that differentiates it from iterative processes.[23] Be that as it may, I have here tried to offer an organised and coherent picture of not only what recursion in fact is, but also of how the recursive techniques Chomsky introduced into linguistics relate to mathematical logic. In particular, I hope this discussion has cleared some of the confusion to be found in the literature, hopefully nullifying the damage done by Watumull, Hauser, Roberts and Hornstein (2014).[24]

[22]There is still the question of the intensional differences among the computational formalisms I have mentioned vis-à-vis the various computational processes cognitive scientists have postulated, but that will have to be discussed in a different publication.

[23]As recurrent processes, both recursion and iteration operate over their own outputs, but in different ways: recursion involves a self-call, and iteration doesn't.

[24]Perhaps unfortunately, WEA is approvingly cited in Hauser et al. (2014), wherein a subset of its authors overlap with those of WEA, but a different subset should really know better (specifically, Chomsky, Berwick, and Yang).

References

Abelson, H. & Sussman, G. J. (1996). *Structure and interpretation of computer programs*. Cambridge, MA.: The MIT Press. (With J. Sussman)

Barendregt, H. (1997). The impact of the lambda calculus in logic and computer science. *The Bulletin of Symbolic Logic*, *3*(2), 181-215.

Blass, A. & Gurevich, Y. (2003). Algorithms: a quest for absolute definitions. *Bulletin of the European Association of Theoretical Computer Science*, *81*, 195-225.

Boolos, G. (1971). The iterative conception of set. *The Journal of Philosophy*, *68*(8), 215-231.

Brainerd, W. S. & Landweber, L. H. (1974). *Theory of computation*. New York, New York: John Wiley and Sons, Inc.

Brookshear, G. (2000). *Computer science*. Reading, MA.: Addison-Wesley.

Buck, R. C. (1963). Mathematical induction and recursive definitions. *The American mathematical monthly*, *70*(2), 128-35.

Chomsky, N. (1956). Three models for the description of language. In *IRE Transactions of Information Theory IT-2* (p. 113-124).

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge, MA.: The MIT Press.

Chomsky, N. (1975). *The logical structure of linguistic theory*. New York, New York: Plenum Press.

Chomsky, N. (1980). *Rules and representations*. New York, New York: Columbia University Press.

Chomsky, N. (1981). *Lectures on government and binding*. The Hague: The Netherlands: De Gruyter Mouton.

Chomsky, N. (1995). *The minimalist program*. Cambridge, MA.: The MIT Press.

Chomsky, N. (2006). *Language and mind*. Cambridge, England: Cambridge University Press.

Chomsky, N. & Miller, G. A. (1963). Introduction to the formal analysis of natural languages. In R. D. Luce, R. R. Bush & E. Galanter (Eds.), *Handbook of mathematical psychology, vol. 2* (p. 269-322). John Wiley and Sons, Inc.

Church, A. (1932). A set of postulates for the foundation of logic. *Annals of Mathematics, Series 2*, *33*, 346-66.

Church, A. (1936). An unsolvable problem of elementary number theory. In M. Davis (Ed.), *The undecidable* (p. 88-107). Dover Publications, Inc.

Church, A. (1941). *The calculi of lambda-conversion*. Princeton, NJ: Princeton University Press.

Collins, J. (2007). Linguistic competence without knowledge of language. *Philosophy Compass*, *2*(6), 880-895.

Collins, J. (2008a). *Chomsky: A guide for the perplexed*. London, UK: Continuum International Publishing Group Ltd.

Collins, J. (2008b). Knowledge of language redux. *Croatian Journal of Philosophy*, *22*, 3-43.

Cutland, N. (1980). *Computability: an introduction to recursion function theory*. Cambridge, England: Cambridge University Press.

Davis, M. e. (1965). *The undecidable*. Mineola, New York: Dover Publications, Inc.

Dean, W. (2007). *What algorithms could not be* (Unpublished doctoral dissertation). Rutgers University.

Epstein, R. & Carnielli, W. (2008). *Computability: Computable functions, logic, and the foundations of mathematics*. Socorro, New Mexico: Advanced Reasoning Forum.

Fitch, W. T. (2010). Three meanings of recursion: key distinctions for biolinguistics. In R. Larson, V. Déprez & H. Yamakido (Eds.), *The evolution of human language* (p. 73-90). Cambridge University Press.

Fitch, W. T., Hauser, M. D. & Chomsky, N. (2005). The evolution of the language faculty: clarifications and implications. *Cognition*, *97*, 179-210.

Fodor, J. A. (1983). *The modularity of mind*. Cambridge, MA.: Bradford Books/The MIT Press.

Gödel, K. (1931). On formally undecidable propositions of the Principia Mathematica and related systems, I. In M. Davis (Ed.), *The undecidable* (p. 4-38). Dover Publications, Inc.

Hauser, M. D. (2009). Origin of the mind. *Scientific American*, *301*(3), 44-51.

Hauser, M. D., Chomsky, N. & Fitch, W. T. (2002). The faculty of language: what is it, who has it, and how did it evolve? *Science*, *298*, 1569-1579.

Hauser, M. D., Yang, C., Berwick, R. C., Tattersall, I., Ryan, M. J., Watumull, J., ... Lewontin, R. C. (2014). The mystery of language evolution. *Frontiers in psychology*, *5*, 1-12.

Hornstein, N. (2009). *A theory of syntax*. Cambridge, England: Cambridge University Press.

Kleene, S. C. (1938). On notation for ordinal numbers. *The Journal of Symbolic Logic*, *3*(4), 150-55.

Kleene, S. C. (1952). *Introduction to metamathematics*. Amsterdam: North-Holland Publishing Co.

Knuth, D. (1997). *The art of computer programming (3 vols.)*. Upper Saddle River, NJ: Addison-Wesley.

Langendoen, T. (2007). Just how big are natural languages? In *Recursion in human languages conference, Illinois State University.*

Langendoen, T. (2010). Just how big are natural languages? In H. van der Hulst (Ed.), *Recursion and Human Language* (p. 139-47). De Gruyter Mouton.

Lobina, D. J. (2011). "A running back"; and forth: A review of *Recursion and Human Language*. *Biolinguistics*, *5*(1-2), 151-69.

Lobina, D. J. (2012a). Conceptual structure and emergence of language: much ado about knotting. *International Journal of Philosophical Studies*, *20*(4), 519-39.

Lobina, D. J. (2012b). *Recursion in cognition: A computational investigation into the representation and processing of language* (Unpublished doctoral dissertation). Universitat Rovira i Virgili.

Lobina, D. J. (2014). *What recursion could not be: a story of conflations.* Manuscript.

Lobina, D. J. & García-Albea, J. E. (2009). Recursion and cognitive science: data structures and mechanisms. In N. A. Taatgen & H. van Rijn (Eds.), *Proceedings of the 31th Annual Conference of the Cognitive Science Society* (p. 1347-1352).

Matthews, R. (2006). Knowledge of language and linguistic competence. *Philosophical Issues*, *16*, 200-220.

McCarthy, J. (1963). A basis for a mathematical theory of computation. In P. Braffort & D. Hirshberg (Eds.), *Computer programming and formal systems* (p. 33-70). North-Holland Publishing Co.

Moschovakis, Y. N. (1998). On founding the theory of algorithms. In H. G. Dales & G. Oliveri (Eds.), *Truth in mathematics* (p. 71-104). Clarendon Press.

Moschovakis, Y. N. (2001). What is an algorithm? In B. Engquist & W. Schmid (Eds.), *Mathematics unlimited: 2001 and beyond* (p. 919-936). Springer.

Moschovakis, Y. N. & Paschalis, V. (2008). Elementary algorithms and their implementations. In S. B. Cooper, B. Lowe & A. Sorbi (Eds.), *New computational paradigms* (p. 81-118). Springer.

Post, E. (1936). Finite combinatory processes. Formulation I. In M. Davis (Ed.), *The undecidable* (p. 288-291). Dover Publications, Inc.

Post, E. (1943). Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, *65*(2), 197-215.

Post, E. (1944). Recursively enumerable sets of positive integers and their decision problems. In M. Davis (Ed.), *The undecidable* (p. 304-337). Dover Publications, Inc.

Pullum, G. K. (2011). On the mathematical foundations of *Syntactic Structures*. *Journal of logic, language and information*, *20*(3), 277-96.

Pullum, G. K. & Scholz, B. C. (2010). Recursion and the infinitude claim. In H. van der Hulst (Ed.), *Recursion and Human Language* (p. 113-138). The Netherlands: De Gruyter Mouton.

Rice, G. (1965). Recursion and iteration. *Communications of the ACM*, *8*(2), 114-115.

Roberts, E. (2006). *Thinking recursively with Java.* Hoboken, NJ: John Wiley and Sons, Inc.

Rodgers, P. & Black, P. E. (2004). Recursive data structure. In V. Pieterse & P. E. Black (Eds.), *Dictionary of algorithms and data structures.* Online at http://www.nist.gov/dads/HTML/recursivstrc.html.

Rosser, J. B. (1939). An informal exposition of proofs of Gödel's Theorem and Church's Theorem. In M. Davis (Ed.), *The undecidable* (p. 223-29). Dover Publications, Inc.

Sieg, W. (1997). Step by recursive step: Church's analysis of effective calculability. *The Bulletin of Symbolic Logic*, *3*(2), 154-80.

Sieg, W. (2006). Gödel on computability. *Philosophia Mathematica*, *3*(14), 189-

207.

Smith, B. C. (2006). What I know when I know a language. In E. Lepore & B. C. Smith (Eds.), *The Oxford Handbook of Philosophy of Language* (p. 941-82). Oxford University Press.

Soare, R. (1996). Computability and recursion. *The Bulletin of Symbolic Logic*, *2*(3), 284-321.

Soare, R. (2007a). Computability and incomputability. In *Proceedings of the Third Conference on Computability in Europe* (Vol. 4497).

Soare, R. (2007b). Incomputability, Turing functionals, and open computing. In *Computation and Logic in the Real World Conference.*

Soare, R. (2009). Turing oracles machines, online computing, and three displacements in computability theory. *Annals of Pure and Applied Logic*, *160*, 368-99.

Tomalin, M. (2006). *Linguistics and the formal sciences*. Cambridge, England: Cambridge University Press.

Tomalin, M. (2007). Reconsidering recursion in syntactic theory. *Lingua*, *117*, 1784-1800.

Tomalin, M. (2011). Syntactic structures and recursive devices: a legacy of imprecision. *Journal of Logic, Language and Information*, *20*(3), 297-315.

Watumull, J. (2013). Biolinguistics and platonism: Contradictory or consilient? *Biolinguistics*, *7*, 301-15.

Watumull, J., Hauser, M. D., Roberts, I. G. & Hornstein, N. (2014). On recursion. *Frontiers in psychology*, *4*, 1-7.

Wirth, N. (1986). *Algorithms and data structures*. USA: Prentice Hall Publishers.