Natural language – no infinity and probably no recursion

Erkki Luuk (erkkil@gmail.com)
Institute of Computer Science, University of Tartu, Liivi 2
50409 Tartu, Estonia

Hendrik Luuk (hendrik.luuk@gmail.com)
Department of Physiology, University of Tartu, Ravila 19
Tartu 50411, Estonia

Abstract[1]

We question the need for recursion in human cognitive processing by arguing that a generally simpler and less resource demanding process – iteration – is sufficient to account for human natural language and arithmetic performance. We claim that the only motivation for recursion, the infinity in natural language and arithmetic competence, is equally approachable by iteration and recursion. Second, we submit that the infinity in natural language and arithmetic competence reduces to imagining infinite embedding or concatenation, which is completely independent from the ability to implement infinite computation, and thus, independent from both recursion and iteration. Furthermore, we show that natural language is a finite rather than infinite set.

Keywords: recursion; iteration; language; brain; infinity; embedding; arithmetic.

Recursion and embedding

An influential line of thought claims that a hallmark of human cognitive processing is recursion (e.g. Fitch, Hauser, & Chomsky, 2005; Hauser, Chomsky, & Fitch, 2002; Premack, 2007). Hauser et al. (2002) drew a distinction between the whole language faculty, including the aspects shared with other species or faculties (the faculty of language in the broad sense) and the unique aspects of the language faculty (the faculty of language in the narrow sense). They hypothesized that the unique aspects of the language faculty comprise "only the core computational mechanisms of recursion as they appear in narrow syntax and the mappings to the Sensory-Motor and Conceptual-Intentional interfaces" (Hauser et al., 2002, p. 1573). Lately, this hypothesis has been vigorously challenged (e.g. (Jackendoff & Pinker, 2005; Pinker & Jackendoff, 2005). Interestingly, none of the challenges question the infinity in natural language, and recursion is contested only in Lieberman (2008) and Bickerton (2009) from a neuroscientific and linguistic perspective, respectively.

[1] An earlier version of this paper "The redundancy of recursion and infinity for natural language" appeared in Cognitive Processing (2011, 12 (1)). As compared to the earlier version, some inaccuracies have been corrected, loosely relevant parts omitted and new arguments added to the present paper.

Recursion

Defining recursion Rogers Jr. (1987, pp. 5-6) gives the following description of a Gödelian recursive definition: "A recursive definition for a function is, roughly speaking, a definition wherein values of the function for given arguments are directly related to values of the same function for "simpler" arguments or to values of "simpler" functions." For example, in the recursive function for defining Fibonacci numbers (for integers n>1), Fib(n) is directly related to Fib(n-1) and Fib(n-2): Fib(n) = Fib(n-1) + Fib(n-2). For the present discussion, the most imporant aspect of recursion lies in its ability to describe an infinity of (input,output) pairs by a finitely definable set of operations in an effectively computable manner (see Odifreddi, 1992, pp. 34-36, for details).

Recursion differs from iteration (another form of repetition) in two essential respects. First, definitions employing iteration do not involve self-reference. Second, without self-reference, every (input,output) pair needs to be defined explicitly, rendering it impossible to define infinite sets by finite means other than by a control flow loop. Computationally, there is a clear difference between a procedure that invokes another instance of itself (recursion) and a procedure that repeats itself either mechanically or with a control flow loop (iteration). Recursion and iteration are the only computational solutions for handling repetition.

As a rule, implementations of recursive functions are slower than those of iterative because recursive functions must allocate memory for their multiple instances. In Abelson et al. (1996), the difference between recursive and iterative **processes** is captured as follows. Recursive processes are characterized by a chain of deferred operations and require that the interpreter keep track of the operations to be performed later on. An iterative process is one whose state can be summarized by a fixed number of state variables, together with a fixed rule that describes how the state variables should be updated as the process moves from state to state. All this being said, recursive **procedures** (or definitions) tend to be formally and notationally more elegant than iterative ones. The difference between process and procedure is explained below.

The influential paper by Hauser et al. (2002) was the first to explicitly formulate the view that recursion is a component of the language faculty (regrettably, no definition of recursion was given in the article). After

reviewing all statements about recursion in the paper the following definition emerges (numbers in brackets refer to the pages in Hauser et al. (2002)): recursion is a neurally implemented (p. 1574) computational mechanism (p. 1573) that yields a potentially infinite array of discrete expressions (pp. 1570, 1571, 1574) from a finite set of elements (p. 1571). Crucially, a computational mechanism with finite input and potentially infinite output, described here, does not imply recursion, as it is possible to generate a potentially infinite output from a finite set of elements without recursive operation by implementing $n \rightarrow \infty$ operations iteratively. Hence, it is not clear whether the mechanism described by Hauser et al. (2002) is recursive or iterative. While it is plausible that the notion of recursion as applied by Hauser et al. (2002) refers to the 'recursion' in the Minimalist Program (Chomsky, 1995), the latter allows for a range of interpretations. Tomalin (2011, p. 308) has, usefully, distinguished between nine different interpretations of 'recursion' in formal sciences, with 'inductive definition' as the most broad one (and thus the safest for syntactic theory). Tomalin's (2011) theoretic variants and computational equivalents of recursion (λ-definability, Turing computability et al.) are more or less on the same level of abstraction but recursion can also be defined in five different levels (in the order of decreasing abstractness):

Table 1. 'Recursion' in five different levels.

1. Inductive (or recursive) definition: A definition with a base rule, specifying that certain "simple" objects are in the class, and an inductive (recursive) rule, stating that if certain objects are in the class, then so are certain other objects formed or derived from them (Minsky, 1972).
2. Recursive definition for a function: A definition wherein values of the function for given arguments are directly related to values of the same function for "simpler" arguments or to values of "simpler" functions (Rogers Jr., 1987).
3. Recursive function: A function that is defined recursively (see 2).
4. Recursive procedure: A procedure that, when executed, invokes another instance of itself (Abelson, Sussman, & Sussman, 1996).
5. Recursive process: An execution of a recursive procedure (see 4).

We suggest that, for cognitive and neural modelling, the procedural level (4) is of central importance. For the following discussion, it is crucial that the recursion we are looking for is implemented in the brain (Hauser et al., p. 1574), i.e. it is not something that is posited at the level of computational theory only (Marr's (1982) level 1 of his three levels of information processing). However, the situation is still very confusing, as it is possible to have neurally implemented recursion of level 1 that is implemented non-recursively (i.e. by iteration) at levels 4-5! In the next section we will give an example of this. It is also important to note that level 1 in Tab. 1 corresponds to Marr's level 1 (computational theory) and levels 4 and 5 in Tab. 1 correspond to Marr's level 2 (algorithm and input/output representation) of information processing. The following section examines whether and how any of the levels in Tab. 1 can be connected to Marr's level 3 (hardware implementation) in the brain.

Recursion, iteration, and inductive definition All open-ended sets (e.g. language expressions, **N**) can be defined inductively, i.e. recursively in the broadest sense. For example, one can have the following inductive definition of 'bear': (a) *Ted is a bear*; (b) *All entities that share at least 98% of Ted's genome are bears*. Observe that, although the set of potential 'bears' is open-ended and inductively defined, no recursion is computationally necessary to determine its contents. An iterative process that compares Ted's genome to that of potential 'bears' would do the job. Importantly, the difference between iteration and recursion pertains to levels 4-5 only. Thus, even if recursion were used in levels 1-3, the involvement of a recursive process or procedure would not be implied, as it can be implemented with a purely iterative computational process (e.g. on a Turing machine).

Below is a strip of iterative pseudocode (1) that defines the infinite set of finite strings ...[X[X[X[XY]]]] that is also defined by the recursive strip (2):

```
(1)  Y → XY (iteration) :
     s=Y                 //assign Y to s
     while true:         //infinite loop:
     rw(Y,XY,s)          //rewrite Y as XY in s

(2)  Y → XY (recursion) :
     s=Y                 //assign Y to s
     rec(s)              //declare function rec(s)
     {                   //start definition of rec(s)
     rw(Y,XY,s)          //rewrite Y as XY in s
     rec(s)              //call function rec(s)
     }                   //end definition of rec(s)
```

As one may observe, (1) and (2) are computationally equivalent – at the level of computational theory, both are described by the rewrite rule Y → XY. Incidentally, this also means that rewrite rules can be "recursive" only in the sense of "recursive definition" (level 1 in Tab 1., which corresponds to Marr's level 1).

From the viewpoint of effective calculability, general recursion and Turing computability are equivalent (Kleene 1952, p. 300), and a universal Turing machine can enumerate any recursively enumerable formal language (an infinite set of finite strings) as can general recursion (Sipser, 1997). Turing machine is based on iteration, whereas general recursion is based only on recursion. Over finite

outputs, recursion and infinite iteration are computationally equivalent (Turing-equivalent).

Crucially, as there is no neural model of recursion (of whatever level), one is unable to identify it in the brain and, accordingly, unable to verify its existence. On the other hand, Lieberman (2008, p. 527) has recently suggested that "neural circuits linking local operations in the cortex and the basal ganglia confer reiterative capacities, expressed in seemingly unrelated human traits such as speech, syntax, adaptive actions to changing circumstances, dancing, and music", thus obviating the need for a neurally implemented recursion. Of course, the distinction between neurally implemented recursive and iterative processes is rather opaque for present-day methods, i.e. the possibility of a neurally implemented recursion cannot be ruled out. We argue for iteration only as a simpler and equipotent computational alternative to recursion.

In sum, there would be no sense in (and no obvious way of) implementing level 1 (in Tab. 1) in the brain separately from levels 4 and 5. As for levels 2 and 3, it would be outlandish to assume that the brain somehow (and apparently redundantly) implements **equations** of the processes it carries out. Thus, implementations of these levels can be ruled out as well. We have also argued that implementations of levels 4 and 5 would be impossible to identify in the brain with present-day knowledge and methods.

Recursion, induction and self-embedding: a confusion In computer science, on the procedural level, recursion denotes the syntactic property that a procedure definition refers to the procedure itself (Abelson et al., 1996). In Chomsky's (1956, 1971 [1957]) phrase structure grammar, recursion is a property of rewrite rules (all that is on the left side of the rewrite arrow repeats on the right side of the arrow, e.g. A → AB) (Chomsky, 1956, 1971 [1957]). Essentially, this notion of recursion reduces to inductive definition. For some other theorists, recursion is a structural property: a situation where an instance of an item is embedded in another instance of the same item (e.g. Heine & Kuteva, 2007; Jackendoff & Pinker, 2005). For clarity, let us call the three recursion, induction and self-embedding, respectively. Recursion and self-embedding are logically independent for the following reasons. First, a self-embedded structure (an NP within an NP, a box within a box etc.) does not have to be recursively generated. Jackendoff and Pinker (2005) submit a picture of a rectangular form within another rectangular form as an example of 'recursion in visual grouping'. Obviously, this has no bearing whatever on recursion. It would be outlandish to assume that recursion is necessary to put a box in a box, or for understanding that a box is in a box. Yet, for conspicuous (but nonetheless insufficient) reasons, this assumption is held with syntactic categories like sentence and NP. The reasons are, of course, the inductive rewrite rules of generative grammar (e.g. NP → A NP), and they are insufficient as the type of induction can be generated iteratively as well (cf. (1)). Furthermore, iteration is defined as the repeated application of a transformation (Weisstein, 2003), which is something that Chomsky's (1956, p. 113) description of his early transformational version of generative grammar explicitly incoporates: "/---/ phrase structure is limited to a kernel of simple sentences from which all other sentences are constructed by repeated transformations". The confusion with recursion can be traced back to Chomsky (1971 [1957], p. 24; 1956, p. 115), who refers to loops as 'recursive devices'. The source that the formalism is taken from refers to the loops as 'circuits' ("a closed series of lines in the graph with all arrows on the lines pointing in the same orientation" – Shannon & Weaver, 1964 [1949], p. 47). The circuits pertain to a graphic representation of finite-state Markov chains, and to call them 'recursive' is jumping to the conclusion, as Markov chains do not prescribe an algorithmic realization for the circuits – Markov chains are confined to Marr's level 1 just like inductive definitions. In Chomsky (1959, p. 143), 'recursive function' is used as a synonym for 'Turing computable function'. Again, this is confusing, as computational equivalence does not imply algorithmic equivalence (which is absent in this case). In sum, the Chomskian notion of "recursion" is a case of confusing Marr's levels of information processing (1 and 2).

Fitch (2010) claims that iterative functions are inadequate for generating center- and/or self-embedding. As an example of the superiority of recursion over iteration, he presents a recursive center-embedding program generating A^nB^n. Below is an iterative pseudocode that does the same:

```
C = ""              //evaluate C to the empty string ""
for i = 0 to n do:          //loop n times
C = concatenate("A",C,"B")    //concatenate "A", C and
                            //"B", and assign the result to C
```

The program embeds *C* between "A" and "B" *n* times, with *i* indicating the depth of embedding in each cycle of the loop. It is true that "recursive functions take their own past output as their next input" (Fitch 2010, p. 75) but this feature is not unique to recursion – in our above examples, concatenate() coupled with iteration does the same.

As for confusing recursion with self-embedding (characteristic to most linguists but not to Chomsky), the two are already in principle very different. Recursion pertains to a process or procedure, self-embedding pertains to a structure. A recursive process or procedure is something that, more often than not, cannot be directly observed. Self-embedding, on the other hand, is usually salient and a subcase of a cognitive phenomenon we term 'hierarchical interpretation'. A defining difference between hierarchical and non-hierarchical interpretation is that only the former allows the same unit to be interpreted simultaneously as a type (i.e. category) and as a token (i.e. instance), hence implying additional interpretative correlates not present in the input. The type/token distinction is a precondition for self-embedding, where tokens are embedded under the same type (e.g. NP or clause). An example of hierarchical

interpretation is natural language. Linguistic interpretation is compounding, merging smaller units that are per se meaningful in the code (Chomsky, 1995; Hauser et al., 2002). As far as we know, linguistic code is unique among natural communication systems in stipulating semantic compositionality, whereby meaningful units are combined into diversely meaningful higher-order units (e.g., words into phrases, sentences and compound words, phrases into sentences and higher-order phrases, etc.).

As an illustration that self-embedding is possible in hierarchical interpretation only, consider the following example: the inductive center-embedding rule AB→AABB generates the strings AABB, AAABBB etc. It is impossible to tell by looking at these strings whether their generation procedure (or process) was recursion, iteration or neither (cf. Fitch, 2010, and the example above). Furthermore, it is impossible to tell whether the strings exhibit self-embedding. Without any a priori assumptions about the generative mechanism (e.g. stipulation of a certain phrase structure grammar), it is undecidable whether a string …AABB... is embedded, concatenated, or elementary (assuming that different generative mechanisms may allow for different elementary strings).

Embedding

Embedding is a situation where an item is embedded in any item (with infinity not implied). Embedding is logically independent from recursion (i.e. there can be one without the other). First, embedding does not have to be generated by a recursive rule. It can be created iteratively or by any other function with relevant output. Second, a recursive process or procedure does not have to yield (relevant) output. Assuming that we cannot witness a recursive process or procedure in situ (e.g. in the brain), two conditions must be met for attesting it: (1) it must generate output, and (2) there must be a one-to-one correspondence between the values of the recursive procedure and its output. Logically, self-embedding is a situation where an instance of an item is embedded in another instance of the same item (with infinity not implied); thus, self-embedding is a proper subset of embedding. The fact that embedding is hierarchical has frequently raised speculations about a putative underlying recursive process or procedure (or more unfortunately, resulted in confusing embedding with recursion). As explained above, a hierarchical or embedded structure is insufficient to decide on its generative mechanism.

Infinity in natural language and arithmetic competence

The central claim of Hauser et al. (2002) and Chomsky (2010) is that a neurally implemented recursive process introduces infinity to natural language and arithmetic. An example of (potential) infinity in natural language and arithmetic competence is the **knowledge** that one can add 1 to n, append a natural language expression to text or embed clauses indefinitely. Of course, we are incapable of **performing** infinitely in any of these tasks (hence the famous competence/performance distinction – Chomsky, 1995).

Chomsky's derivation of neurally implemented recursion for operating on **N** is as follows. A) Any formal definition of the set of natural numbers **N** incorporates recursion by means of the successor function, where 1 = S(0); 2 = S(S(0)) etc. B) We have knowledge of the properties of **N** (i.e., given enough time and space, we can compute the sum of two numbers, and distinguish the right from the wrong answer). From premises A and B he conjectures that neurally implemented recursion is required for operating on **N** (e.g. for adding 4555 to 7884). Thus, we have the following: (a) infinity in natural language and arithmetic competence (to motivate neurally implemented recursive process in the first place), (b) neurally implemented recursion is required to operate on **N**, and (c) **N** is an offshoot of the language faculty. From these premises, it follows that (d) neurally implemented recursion underlies both **N** and natural language.

On the face of it, the above argument for neurally implemented recursion is consistent and logically sound. However, premises (a)-(b) are false, and in section "Arithmetic performance" we argue in more detail that neurally implemented recursion is not necessary to operate on **N**. As explained below, infinity in natural language and arithmetic competence reduces to **imagining** infinite embedding or concatenation, and thus does not qualify as an output of a recursive process or procedure (as there is no reason to assume that conceptualizing infinity requires recursion). The concept of neurally implemented recursion is largely motivated by the 'discrete infinity' property of natural language (Chomsky, 1995; Hauser et al., 2002). In fact, the whole distinction between the broad and the narrow language faculties, as originally proposed by Hauser et al. (2002), can be derived from this property. Importantly, the infinity of natural language has been always taken as axiomatic and never proven. The last instance that Chomsky appeals to in this question is Wilhelm von Humboldt (1999 [1836]) who simply states the infinity of natural language as a fact.

One might start from the observation that the maximum possible natural language "corpus" – everything that has ever been and will be processed – is not infinite but a finite, just physically uncountable set. We propose that this is precisely the nature of language as it should be accounted for. In fact, the very spacetime that can support physical computational systems is finite (Krauss & Starkman, 2000). This substantial correction (physically uncountable finity instead of infinity) is suggested for the sake of unambiguity and exactitude. Physically uncountable sets can be finite or infinite. Set-theoretically, potential and actual infinity (Moore, 1990) are proper subsets of physical uncountability. The evidence that Hauser et al. (2002, p. 1571) submit for discrete infinity covers also physically uncountable finity: "There is no longest sentence (any candidate sentence can be trumped by /---/ embedding it in "Mary thinks that ...)". It

would be a contradiction to assume that the size of a finite, physically uncountable array can be compared to the size of all others, or that such array can be embedded (the mere fact that we can imagine embedding such an array does not account for its capacity of being embedded). "There is no non-arbitrary upper bound to sentence length." This is as true for an infinite as it is for a finite, physically uncountable array.

The finity of natural language can be also derived logically, without invoking physically instantiated computation:

1. Natural language has a limit which is either infinite or finite.
2. Natural language computation takes time.
3. From 1 and 2 it follows that, for any given moment in time, there is an infinite number of finite limits that are never reached.
4. Assuming that the cardinalities of natural language and **N** are equal[2], there is only one infinite limit that is never reached.
5. For any finite limit that is never reached, the probability of natural language having it is > 0.
6. From 3-5 it follows that the probability of natural language having the infinite limit is 0.

Arithmetic performance

If recursion were involved in conceptualizing numbers, our brain would execute something like a successor function …S(S(S(0)))… for natural numbers and maybe also n*n*n*n… for base-n integer exponents (since we normally use base-10 numeral system, n would normally equal 10). While n*n*n*n… can be coded and implemented recursively as well as iteratively, it is unlikely that anything approximating …S(S(S(0)))… or n*n*n*n… would be run in our brains for conceptualizing numbers and performing arithmetic on them. If it were, our arithmetic performance should be significantly better than it tends to be. If, on the other hand, our inferior arithmetic skills are down to general performance limitations and/or penalties for (other) arithmetic operations, there would be no apparent use for running these procedures for our mathematical capacity. The only remaining justification for …S(S(S(0)))… and n*n*n*n… would be recursion in the language faculty. However, for this concession to make sense, there would first have to be some evidence for recursion in the language faculty. As we have argued at length above, at present we have merely conjectures built on invalid premises (see section "Infinity…").

It is easy to demonstrate that conceptualizing a principle (recursion) for producing a pattern of output (**N** or self-embedding) does not entail (1) that the principle is necessary for producing the pattern (as **N** or self-embedding can be also produced iteratively), and (2) that the principle itself must be neurally implemented for us to be able to conceptualize it. For example, we can conceive that all natural numbers are derived from the number 20098 by +/-1 operations, i.e. each time we conceptualize a natural number x that is less than 20098, we subtract 1 from 20098 until we get x and each time we conceptualize a natural number y that is greater than 20098, we add 1 from 20098 until we get y. We can conceive this principle. Does it follow that the "20098 +/-1" principle must be neurally implemented for us to be able to conceive it in the first place? Surely not. Observe that the situation with the "20098 +/-1" principle is similar to the recursive one: we can conceptualize the principles but both are at odds with human arithmetic performance. To circumvent the latter problem in the neurally implemented recursion hypothesis, the competence/performance distinction has been called into effect. However, a competence/performance distinction could be also invoked for explaining why our performance is at odds with the "20098 +/-1" principle. Besides, the distinction raises a non-parsimonious psychological duality as to the conceptualization of relevant syntactic and arithmetic properties – we can conceptualize that the properties are given to us by a recursive principle, but we do not seem to follow the principle neither in linguistic nor arithmetic processing/performance. Furthermore, it seems inconsistent to explain the apparent discontinuity between competence and performance in arithmetic and language by e.g. limitations in primary memory – what potential advantage could a neurally implemented recursive principle bestow if its effects are subject to so severe constraints?

Conclusion

We conclude with the following points. First, both recursion and iteration allow for finite definitions of infinite sets. Moreover, iterative solutions are frequently less resource demanding than recursive ones (cf. section "Defining recursion"). Second, three logically independent notions of "recursion" are being conflated and confused in linguistics (e.g. Chomsky, 1956, 1971 [1957]; Heine & Kuteva, 2007; Jackendoff & Pinker, 2005): (A) recursive algorithm (Marr's level 2), (B) recursive (or better, inductive) definition (Marr's level 1), and (C) an instance of an item embedded in another instance of the same item. We suggest a terminological way out of the confusion, by reserving 'recursion' for (A) for which there are no alternative terms, and designating (B) and (C) 'induction' and 'self-embedding', respectively. Third, the technical preciseness of the notion of recursion makes it next to impossible to find evidence for it in the brain with the present-day methods, and there is no reason to assume neurally implemented recursion by default (see below). Fourth, contrary to Chomsky (Chomsky, 1995; Hauser et al., 2002) and many others, we argue that a property of natural language is not discrete infinity but physically uncountable finity. Fifth, we

[2] Since we are interested in an upper bound of computation/processing time, $\aleph_0$ is sufficient. It is difficult to develop the argument here but the very fact that time intervals seem to exist suggests that time is not infinitely divisible ($\aleph_1$ and beyond). Other indications of this are e.g. Achilles and the tortoise paradox and Planck time.

reject the received opinion, articulated by Chomsky et al. (Chomsky, 2010; Fitch et al., 2005; Hauser et al., 2002), that neurally implemented recursion is necessary to explain natural language and arithmetic competence and performance. The only motivation for neurally implemented recursion is infinity in natural language and arithmetic competence (e.g. the knowledge that one can add 1 to n, append a natural language expression to text or embed clauses indefinitely). We claim that infinity in natural language and arithmetic competence reduces to imagining infinite embedding or concatenation, which is completely independent from an algorithmic capacity for infinite computation, and hence, completely independent from neurally implemented recursion or iteration. In sum, there is no infinity in natural language and arithmetic processing, but even if there were, iteration would be sufficient for generating it.

Acknowledgments

We thank Noam Chomsky and Margus Niitsoo for extremely helpful, thorough and critical discussions, and Märt Muts, Lameen Souag, Lutz Marten, Tania Kuteva, Michael Corballis, Panu Raatikainen, Tim Gentner, Geoffrey Pullum and the anonymous reviewers for their comments and suggestions. All remaining mistakes are our own. Erkki Luuk was supported by the target-financed theme No. 0180078s08, the National Programme for Estonian Language Technology project "Semantic analysis of simple sentences 2", the Alexander von Humboldt Foundation, and the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

References

Abelson, H., Sussman, G. J., & Sussman, J. (1996). *Structure and Interpretation of Computer Programs* (2nd ed.). Cambridge, MA: MIT Press.

Bickerton, D. (2009). Recursion: core of complexity or artifact of analysis? In T. Givón, & M. Shibatani (Eds.), *Syntactic Complexity: Diachrony, Acquisition, Neuro-cognition, Evolution*. Amsterdam: John Benjamins.

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

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, *2*, 137-167.

Chomsky, N. (1971 [1957]). *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1995). *The Minimalist Program*. Cambridge, MA: MIT Press.

Chomsky, N. (2010). Some simple evo-devo theses: how true might they be for language? In R. K. Larson, H. Yamakido, & V. Deprez (Eds.), *Evolution of Human Language: Biolinguistic Perspectives*. Cambridge: Cambridge University Press.

Fitch, W. T. (2010). Three meanings of "recursion": key distinctions for biolinguistics. In R. K. Larson, V. Deprez, & H. Yamakido (Eds.), *The Evolution of Human Language: Biolinguistic Perspectives*. Cambridge: Cambridge University Press.

Fitch, W. T., Hauser, M. D., & Chomsky, N. (2005). The evolution of the language faculty: clarifications and implications. *Cognition*, *97*(2), 179-210; Discussion 211-125.

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*(5598), 1569-1579.

Heine, B., & Kuteva, T. (2007). *The genesis of grammar: a reconstruction*. New York: Oxford University Press.

Jackendoff, R., & Pinker, S. (2005). The nature of the language faculty and its implications for evolution of language (Reply to Fitch, Hauser, and Chomsky). *Cognition*, *97*(2), 211-225.

Krauss, L. M., & Starkman, G. D. (2000). Life, the universe, and nothing: Life and death in an ever-expanding universe. *The Astrophysical Journal*, *531*, 22-30.

Lieberman, P. (2008). Cortico-striatal-cortical neural circuits, reiteration, and the "narrow language faculty". *Behavioral and Brain Sciences*, *31*, 527-528.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Fransisco: W. H. Freeman and Company.

Minsky, M. (1972). *Computation: Finite and Infinite Machines*. London: Prentice-Hall International.

Moore, A. W. (1990). *The Infinite*. London: Routledge.

Odifreddi, P. (1992). *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers* (Vol. 125). Amsterdam: Elsevier.

Pinker, S., & Jackendoff, R. (2005). The faculty of language: what's special about it? *Cognition*, *95*(2), 201-236.

Premack, D. (2007). Human and animal cognition: Continuity and discontinuity. *Proceedings of the National Academy of Sciences of the United States of America*, *104*(35), 13861–13867.

Rogers Jr., H. (1987). *The Theory of Recursive Functions and Effective Computability*. Cambridge, MA: MIT Press.

Shannon, C. E., & Weaver, W. (1964 [1949]). *The Mathematical Theory of Communication*. Urbana: The University of Illinois Press.

Sipser, M. (1997). *Introduction to the Theory of Computation*. Boston, MA: PWS Publishing Company.

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

Weisstein, E. W. (2003). *CRC Concise Encyclopedia of Mathematics* (2nd ed.). Boca Raton: Chapman & Hall/CRC.

von Humboldt, W. (1999 [1836]). The diversity of human language-structure and its influence on the mental development of mankind, *Wilhelm von Humboldt: On Language*. Cambridge: Cambridge University Press.