Recursion and Cognitive Science: Data Structures and Mechanisms

David J. Lobina (davidjames.lobina@urv.cat)
CRAMC, Department of Psychology (URV)\Department of Philosophy (UB)
Ctra. de Valls s\n, 43007, Tarragona, Spain

José E. García-Albea (jegarcia.albea@urv.cat)
CRAMC, Department of Psychology, Ctra. de Valls s\n
43007, Tarragona, Spain

Abstract

The origin and application of Recursion in the formal sciences is described, followed by a critical analysis of the adoption and adaptation of this notion in cognitive science, with a focus on linguistics and psychology. The conclusion argues against a widespread mistake in cognitive science, and recommends recursion should only be used in reference to mechanisms.

Keywords: Recursion; Hierarchy; Iteration; Combinatory Operations; Data-structures.

Introduction

Recursion has been recently identified as the defining feature not only of natural language (Hauser, Fitch, & Chomsky, 2002), but of human cognition overall (Corballis, 2007). However, it has received less than a satisfactory characterisation. More often than not, it has been applied to the structural complexity of some of the representations the human mind seems to have and use, irrespective of the mechanisms operating over these representantions. This is in clear discrepancy with the formal sciences, where recursion originated. The following section identifies its origin and describes its employment in mathematical logic and computer science. Subsequent sections provide critical descriptions of its adoption in cognitive science, with particular attention to linguistics and psychology. The last section concludes the essay.

Recursion and the Formal Sciences

The word "recursion" entered the English language in the 17th century as an adaptation of the past participle of the Latin verb "recurrere". It meant 'a running back, backward course, return' and was used in this sense by Robert Boyle in his *New Experiments Physico-Mechanical*: 'the recursions of that Pendulum which was swinging within the Receiver'.

By the early 20th century this use was 'rare, and obsolete' and was so recorded in the 1933 edition of the *Oxford English Dictonary*. Concurrently, however, mathematicians were starting to use this term in a different, more technical, manner.

Recursion and Mathematical Logic

The more technical designation made reference, as Soare (1996) shows, to a 19th century technique: definition by induction[1]. Thus, a function is recursive if it's defined by induction; that is, if it's defined in terms of itself: the characteristic feature is that each value is specified in terms of previous values that the same function has already calculated.

[1]We here focus on the original interpretation only —see Soare's paper for more details.

Below follows a recursive definition of the class of factorials, i.e., $n! = n \times n-1 \times n-2 \ldots \times 3 \times 2 \times 1$, where n is a natural number:

Def. $n!$
 if $n = 1$, $n! = 1$ (base case)
 if $n > 1$, $n! = n\times\ (n-1)!$ (recursive step)

The calculation of the factorial of, for instance, 4 (i.e., $4! = 4\times\ 3!$), sees the factorial function calling itself in order to calculate the factorial of 3, and so on until it reaches the factorial of 1, the base case, effectively terminating the recursion."Self-reference" is therefore the prominent feature, making recursion particularly apt to define infinite sets[2].

Recursion is also used in what mathematicians call "inductive proofs", which are usually employed to prove if a given property applies to, for example, every natural number. It proceeds as follows: first we show that a given statement is true for 1, then we assume it is true for n, a fixed number (the inductive hypothesis), and finally we show it's therefore true for $n+1$ (the recursive step). If every step is followed correctly, we conclude the statement is true for all numbers. An inductive proof *employs* recursion, but it should not be confused with it proper. After all, a recursive definition involves no "inductive hypothesis".

Recursion and Computer Science

Computer scientists' employment of recursion is similar, although it differs in subtle but significant ways. This seems to follow from the different aims these two disciplines have[3].

It appears to be customary to describe what mathematicians study as "declarative knowledge", the "what is", while computer scientists focus on "imperative (procedural) knowledge", the "how to". Consequently, computer scientists are under a number of constraints (inter alia: memory limitations, computational complexity, etc.; i.e., efficiency) that mathematicians don't seem to worry too much about. Specific re-

[2]Cf.: '[t]he power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement' (Wirth, 1986, p. 136).

[3]This section draws heavily from Abelson and Sussman (1996).

percussions for the role of recursion therein follow, but some brief definitions must precede their description.

Computer "procedures" describe the rules for manipulating data; they are much like mathematical functions, with the difference that the former must be effective. Each datum is called an "expression", which can either be a primitive element, or two or more elements linked by an operator. We interact with a computer language by typing expressions via an "interpreter", which "evaluates" these expressions. Every evaluation follows the basic cycle: *a*) evaluate all the elements of an expression; if it's a compound, evaluate all the elements of the subexpressions; and *b*) apply the procedure of the operator to the operands.

The first step establishes that in order to evaluate a complex expression, the interpreter must evaluate each element of the subexpression first. The evaluation rule contains therefore an invokation of the rule itself; it's a recursively-defined procedure.

More significantly, computer scientists have studied what it means for a "process"(the rules of manipulation procedures describe) to proceed recursively. Take the recursive definition of the factorial functions from the precedent section. The following is the process executed by this procedure to calculate the factorial of 4:

(factorial 4)

(4 $\times$ (factorial 3))

(4 $\times$ (3 $\times$ (factorial 2)))

(4 $\times$ (3 $\times$ (2 $\times$ (factorial 1))))

(4 $\times$ (3 $\times$ (2 $\times$ 1)))

(4 $\times$ (3 $\times$ 2))

(4 $\times$ 6)

24

As the shape of the process reveals, there is an expansion followed by a contraction, as chains of deferred operations are built up before they are orderly performed —some sort of memory is needed to keep track of these operations. It's this accumulation of unfinished tasks, the result of a given operation calling itself (self-reference), that characterises recursive processes.

There exist alternative ways to calculate factorials, though. A related and relevant method starts by multiplying 1 by 2, then the result by 3, then by 4, until we reach *n*. The process keeps a running product together with a counter from 1 up to *n*, and the stipulation that *n*! is the value of the product when the counter exceeds *n* finalises it. The resulting process is "iterative":

$$\begin{array}{llll}(factiter & 4 & 1 & 1) \\ (factiter & 4 & 2 & 1) \\ (factiter & 4 & 3 & 2) \\ (factiter & 4 & 4 & 6) \\ (factiter & 4 & 5 & 24)\end{array}$$

The first digit (4) indicates the number whose factorial we want to compute. The second and third represent the counter and the product, respectively, and both start at 1. The process proceeds thus: the product is multiplied by the counter, then the counter is increasing by 1, the previous product is now multiplied by the new counter, and so on until the counter reaches a value higher than the number whose factorial the process is calculating. The product at that point is the result of the operation.

Recursion and iteration are closely related. In fact, they are both types of recurrence: both involve the repetition of an operation (and both need the establishment of termination conditions). The former, however, involves self-reference and as a result chains of unfinished tasks, placing a heavy burden on memory. In the case of the latter, the state of an iterative process can be summarized at any stage by the number of variables plus the fixed rule that establishes how the variables are updated from one state to another. This is not possible with a recursive process, as the state at any stage must take into consideration the deferred operations stored in memory. In general, this makes iteration more computationally efficient, but certain tasks naturally call for a recursive rather than an iterative process.

The recursive method employed to compute factorials was based on the rather subtle observation that we could solve the problem by reducing it to one or more subproblems identical in structure but simpler to solve (Roberts, 2006, p. 4). A problem must in fact meet three properties in order to be recursively solved: *a*) the original problem must be decomposable into simpler instances of the same problem, *b*) the subproblems must be so simple that they can be solved without further subdivision; and *c*) it must be possible to combine the results of solving these subproblems into a solution to the original problem (Roberts, 2006, p. 8). Consequently, the description of any recursive solution —the procedure— must be general enough so that it applies to the original problem and any subproblems; it must be able to call the original method with new arguments as it proceeds.

Our recursive solution to compute factorials met all the criteria, but it took a subtle observation to do so. That is, even though we managed to divide the original problem into hierarchically organised subtasks, there was prima facie no sign of such hierarchy in the data themselves. There are tasks, however, where an internal hierarchy is evident and a recursive strategy is in principle the most natural solution.

Paradigmatic cases include problems, functions or data structures that are already defined in recursive terms (Wirth, 1986, p. 135). By a recursive data structure, we understand, following the definition of the U.S. National Institute of Standards and Techonology, an object or class 'that is partially composed of smaller or simpler instances of the same data structure'[4]. That is, a structure that includes an abstraction of itself, and "trees", "lists" and the like constitute the prototypical cases —trees inside other trees, or lists inside lists.

[4]URL: http://www.itl.nist.gov/

This introduces a distinction between "structural recursion" and "generative recursion" and much work has been devoted to work out how close the fit between the two is. That is, do recursively-defined structures always call for recursive mechanisms to operate over them? The answer is yes in object-oriented programming languages, as the form of the data establishes the form of the algorithm, but it is not automatic in other cases. In general, there is in fact no guarantee that a recursive algorithm is the best way to solve a recursively-defined problem, as iteration might be a better option.

There is therefore a natural fit between hierarchy and recursion, but it's a matter of research to work out if the orbiting conditions of each problem suggest recursion or iteration.

Recursion and Cognitive Science

Ever since the advent of the representational-computational paradigm, the formal sciences have informed the study of cognition to a great extent, and recursion hasn't been an exception. More often than not, however, scholars have exclusively focused on the data structures the mind seems to have and use, rather than on the mechanisms operating over them. This clearly departs from the formal sciences, and this shift hasn't really been properly justified.

The following subsection provides examples from linguistics, where its treatment has been the most prominent. Following subsections briefly describe possible applications in other domains, such as higher-order cognition, or central systems, and the final section ends the essay with a set of concluding remarks.

The analysis will focus on two of the three levels of explanation Marr (1982) outlined: *a*) the computational, which provides an abstract characterization of the domain under study, focusing on the mapping between one type of information into another, including the abstract properties that derive, and *b*) the algorithmic, which focuses on how this mapping is actually performed —it attempts to determine the appropriate representation of the input and output, and the algorithm employed for the transformation.

The following subsections implicitly support a Classical architecture, where both representations and computations are in fact postulated. It is for others to establish how any of the following bears for a connectionist outlook, but no implications can be legitimately derived from what we say.

Recursion and Linguistics

Work under this epigraph is an example of Marr's computational level, and recursion has featured therein ever since the 1950's. Emile Post's "rewrite rules" were at the time used to account for how sentences were generated:

(a) $S \rightarrow NP\ VP$

(b) $NP \rightarrow D\ N$

(c) $VP \rightarrow V\ NP$

(d) $NP \rightarrow NP\ PP$

(e) $VP \rightarrow V\ S$

The last two rules are recursive, as categories to the left of the arrow are reintroduced on the right hand side. The recursion in (d) is direct: the NP rule rewrites NP as a result. In (e) the recursion is indirect: an S rule, (a), generates NP and VP, and (e), a VP rule, reintroduces S. These rules can generate embedded structures such as 'The guy [with the green car]', and 'John thinks (that) [Michael killed the policeman]', respectively. Recursion then applied to the rules of formation, much like in the formal sciences. However, since the advent of the Minimalist Program (MP; Chomsky, 1995), one single operation is postulated: Merge. The role of recursion has varied since then, and while some scholars employ it to characterise the operations of Merge, the majority focus on the structures Merge purportedly operates over. We provide below critical descriptions of recent examples of both views.

Recursive Structures Pinker and Jackendoff (2005, PJ) define recursion as, firstly, a 'procedure that calls itself', but then add it also applies to 'a constituent that contains a constituent of the same kind' (p. 203); self-embedding. We provided examples of self-embedding above, which were generated by clearly recursive rules in the 1950's. However, PJ talk of recursive structures independent of the mechanisms that generate them, and this is rather widespread in the literature. Boeckx and Uriagereka (2007), for instance, point out that embedding is responsible for the recursive characteristics of language, but provide no description of the recursiveness of the mechanisms responsible for the generation of embedded structures.

Much can be said about PJ's definition. 'A constituent inside a constituent of the same kind' has *kind* refer to the element heading a phrase, making an NP inside another NP a case of self-embedding, but an NP inside a VP not quite. Yet, one of the fundamental results in linguistics has been the discovery that *all* phrases (NPs, VPs, etc.) have the same configuration: an asymmetric [Spec [Head - Comp]] structure, regardless of which element heads the phrase. In this respect, all structures manifest self-embedding: a sentence is ultimately a collection of hierarchically organised phrases inside phrases of the exact, same geometrical shape. Nesting is therefore the most prominent feature of tree representations of sentences, and recursive mechanisms would in principle care little about which element heads each phrase.

And yet for the most part linguists talk of recursion with no reference to mechanisms. Neeleman and Van de Koot (2006), for instance, identify recursion with nested structures: 'recursion will result if there is a set of primitive trees that can be combined into a structure in which the root node is repeated in the yield' (ft. 5, p. 1530). We are not told how these primitive trees are actually combined, so it's impossible to establish if the combination is recursive or not — i.e., if recursion "results". The onus seems to be on the structures that are repeated, that recur, as Hinzen (2008) explicitly states: 'it is only particular domains of syntactic organization that productively "recur" at all' (p. 359). *Recur* is a rather poor choice of words; *Recurse* would be the right word to use, but there's no indication he has anything other than recurrence in mind.

This is not an isolated case; many other scholars use the term recursion to actually mean recurrence. Medeiros (2008) contains myriad mentions of recursion, recursive, etc., and an early appearance has it that 'certain consistent patterns in recursion' amount to 'repeated structural "templates"' (p. 153). Immediately after, the phrase "recursive templates" substitutes "repeated structural templates", and the rest of the paper analyses the geometrical properties of syntactic trees, with an emphasis on "the minimal template structure" (perhaps part of the primitive trees mentioned above), apparently needed to 'have recursion at all' (p. 174). The phrase "recursive shape" appears throughout the paper, but it refers to the geometrical shape of nested tree represetantions.

The obvious point to make is that recursion is being applied to structures even if no indication is given as to how any of these are actually generated. And yet, this is not the situation we find in the 1950's when recursion was first employed in linguistic studies.

Recursive Mechanisms In the 1950's recursion was used in reference to mechanisms: 'the output (of the language acquisition device) is a system of recursive rules' (Chomsky, 1967, p. 455). As Tomalin (2007) shows, Chomsky argued the need for a grammar to have 'recursive steps' as early as 1956; 'recursive devices' were needed, it was then argued, in order for the grammar to be able to 'produce infinitely many sentences'. It has been 'conventionally assumed', as Tomalin puts it (2007, p. 1785), that recursive components allow a grammar to generate a potentially infinite set of syntactic structures. This is however not reason enough to employ recursion, as iteration may suffice. One needs to add that syntactic structures are hierarchically organised, which certainly calls for a recursive mechanism. A few examples will illustrate[5].

The introduction to Maratz, Miyashita, and O'neil (2000), for instance, states that the 'combination of units in language is recursive' (p. 3), but the actual example they use actually isn't. According to them, the derivation of *The man saw the cat* starts by merging *the* and *cat*, 'the result of (this) combination becomes a unit for further combination, here with the word *saw*' (ibid.). As described, the operation is strictly recurrent, but not necessarily recursive. Things don't improve if we add, following Chomsky (1995, p. 248), that Merge 'embeds (an object) within some construction ... already formed'. That merely makes Merge an operation that embeds elements into other elements, but it doesn't follow it's recursive. In order for Merge to apply recursively, the derivation must contain chains of deferred operations.

[5]Tomalin (2007) analyses how recursion was employed in mathematical logic and its influence in syntactic theory. However, he doesn't consider its employment in computer science and its potential influence in cognitive science. He therefore focuses on how to recursively define the operations of Merge (the procedure), but not on whether the process his recursive definition describes proceeds recursively (see supra).

Di Sciullo and Isac (2008) outline the operations of Merge in some detail, which allows us to see if it applies recursively. However: 'Merge is *recursive*, where the output of Merge may subsequently be submitted to Merge with other elements yielding a further constituent' (ibid., p. 261). This definition is incorrect: it merely describes a recurrent operation.

Their description of Merge merits some attention, though. A derivation starts with a Numeration —a list of lexical items out of which Merge will yield a syntactic structure. There are two types of Merge: External (EM; it takes two objects from the Numeration and merges them) and Internal (IM; it takes an element from an already-built object and *moves/copies* it to a different location in the structure it's constructing); both can only operate over two objects at a time. As they describe it, EM 'iteratively selects items from the numeration, one by one, until the numeration is exhausted and a complex object is formed that contains all of the items that started out as individual elements' (Di Sciullo & Isac, 2008, p. 261). Quite right: as there are no deferred operations, EM is clearly not recursive, even if they seem unaware of the discrepancy.

IM is another matter, though. As soon as EM introduces an object that will be moved/copied to another location later on by IM, a chain containing a deferred operation is created. It then seems that recursion is a property of IM only, which it has surprisingly only been mentioned in passing: Epstein and Hornstein (2000) mention that recursion is 'relegated to the transformational (i.e., movement) component' (p.xii), while Soschen (2008) states that a 'relation between individuals may constitute a phase and induce movement (recursion)' (p. 212).

Thus, recursion appears to apply in the operations of IM only, but it's not clear how promiscuous a property of IM it is. This depends on a number of things: how many elements triggering movement are introduced in the derivation, when they are introduced and in which order the movement/copying operations apply. Movement may well apply iteratively rather than recursively if these orbiting conditions don't create chains of deferred operations.

Coda The employment of recursion in linguistics clearly departs from the formal sciences in one respect: the focus lies on data structures, rather than on the mechanisms these structures call for. As Roberts (2006, p. 7) puts it: 'one needs to recognise that "recursiveness" is a property of the *solution* to a problem and not an attribute of the problem itself'.

As a caveat, one could also talk of "structural" and "generative" recursion, but why multiply terms unnecessarily? When referring to "recursive structures", linguists have either self-embedding, nesting, or simply recurrence in mind, so why not use those terms instead? Further, some structures described as recursive may well not be generated recursively, an inconsistency that ought to be avoided.

Psycholinguistics

Most studies of syntactic processing have focused on how new material is added to the already-built structure as the parser proceeds (i.e., where it is attached), but no-one has so far probed if the parser's structure-building mechanism, for there must be one, proceeds recursively or not[6].

Some factors suggest recursive processing to be at least possible. As a sentence, a TP (or else), is an asymmetric phrase structure composed of other asymmetric structures (NPs, VPs, etc.), we are then faced with a complex problem (processing a TP), easily reducible to simpler instances of the same problem (processing NPs, VPs, PPs, in succession); a recursive solution is prima facie ideal.

Parsing a sentence therefore involves building a TP phrase, but its completion depends upon the completion of every internal phrase. Both a recursive and a non-recursive strategy can in principle solve the same task; thus, figuring out the two strategies would allow us to predict how subjects would proceed at any given moment. Different types of structures, leading to different predictions, could then be devised and tested[7].

Recursion and General Cognition

Much like in linguistics, many cognitive scientists have also applied recursion to both data structures and mechanisms, and some of the same problems arise. We'll provide a few examples of both cases, and will close the essay with the conclusions.

Recursive Structures Corballis (2007) illustrates a few examples of what he takes to be recursion in different domains. He starts with language, and much like the previous section, focuses on nested structures, which turns out to be the main feature of his entire discussion. Other examples include Theory of Mind abilities (i.e., belief ascription), which he argues involve recursion. He divides these abilities into two levels: *a*) zero-order theory of mind, i.e. mental processes such as thinking or knowing; and *b*) first-order, i.e. thinking or knowing what *others* are thinking or knowing, which involves recursion.

The latter is implied in statements like "Mark thinks Lauren thinks he is an idiot". Some experimental evidence suggests that children's abilities in understanding self-embedded sentences and self-embedded beliefs/desires is almost concurrent (P. H. Miller, Kessel, & Flavell, 1970), even if the replication of these experiments shows that comprehension of "recursive ToM" abilities starts a bit earlier (Oppenheimer, 1986). Still, the experiments tells us nothing about how children *represent* self-embedded beliefs/desires, and even less about how children *process* self-embedded sentences OR self-embedded beliefs/desires. In short, the role of recursion within is not clear.

The same applies to Corballis's other examples, like episodic memory abilities (e.g., "I know I experienced X"), or in the apparent hierarchical conceptualization of tool making. All these examples make reference to hierarchically nested structures, and its application can be as broad as we can imagine.

It is clearly the case that a basic feature of cognition is that our thoughts can be composed of different bodies of information from different modalities. There must be a conceptual system where it all comes together, and this system is likely to contain thoughts within thoughts. The overall problem remains though: it's one thing for the representations the mind has and uses to exhibit hierarchy and nesting, but it's another thing completely for the mental processes operating over these representations to apply recursively.

Recursive Mechanisms Much work has been undertaken in the study of the architecture of cognition, and as Pylyshyn (1984, 1988) points out, a computational system, whetever the actual details, is one that reads, writes and transforms structured representations.

Let us focus on the system's component in charge of ordering the rules/operations that apply over the data structures: the Control. Control is in charge of sequencing action, but not only from point to point, as it can also transfer control to another locus, therefore creating subroutines. Subroutines can send control to other subroutines, and a hierarchy of nested operations develops. Once each subroutine is completed, control is sent back up to where it was transferred from, and so on until it reaches the highest Control operation.

G. A. Miller, Galanter, and Pribram (1960) were perhaps the first scholars to outline a detailed model of serially-ordered compositional systems, and the focus was on TOTE (test-operate-test-exit) units, which they argued were the basic Control unit of cognition.

This operation is based on feedback loops (self-reference), and TOTEs can therefore be nested into other TOTEs (chains of deferred operations therefore arise), making it ideal for solving complex tasks divisible into functionally-equivalent but simpler subtasks. It's been recently described as an instantiation of the Standard Account in early cognitive science (Samuels, forthcoming) —a "plan, then execute" model of behaviour— and it's ideally suited to account for the hierarchical organization of the cognitive architecture. In short, it constitutes the clearest case of recursion in cognition.

Conclusion

Recursion developed in the formal sciences in reference to combinatory operations, mechanisms and the like.

In the 1950's, linguists correctly employed recursion in reference to specific rewrite rules, but ever since their elimination from linguistic theory, most linguists have used recursion, rather puzzlingly, to refer to those structures that recursive rewrite rules were used to generate. This may well be the unfortunate legacy of employing rewrite rules. Other linguists

[6]Mention of recursion in the psycholinguistics literature usually refers to self-embedding (central or tail), a related but entirely different matter to what we're saying here.

[7]This empirical work is currently being undertaken by the authors.

have focused on the operation Merge, which is welcome, but a satisfactory treatment is yet to be provided.

There seems to be a strong tendency to confuse hierarchically-structured representations with recursion. Even though hierarchical data structures call for recursive mechanisms, the latter are not automatic because of the former. Recursion always involves hierarchy, but not all hierarchy involves recursion —iteration may well be the right candidate for some structures/tasks. Since all computational tasks that can be solved recursively can also be solved iteratively, extra care needs to be employed when arguing for one or the other.

There are however good reasons to believe that recursive mechanisms do apply in cognition, and brief descriptions of these mechanisms have been provided. Most of our discussion has focused on Marr's computational level (i.e., Merge and TOTE units), as little work on recursive processing has been undertaken in cognitive science. Some indications have been provided of what this empirical work might actually involve, and this short essay may well be regarded as a theoretical clean-up of the role of recursion in cognitive science.

It is not a matter of terminology; applying recursion to structures or to mechanisms, or to the fit between the two, results in substantial claims regarding the specific properties of the representations and operations the mind manifests. It is in this light that most statements regarding the uniqueness of recursion in either human language or cognition should be viewed. There are strong reasons to think that hierarchically-structured representations are unique in humans, but it is no small matter to discover if the mechanisms operating over them proceed recursively or not, with all the related issues it involves (memory load, architectural complexity, etc.).

It is therefore strongly recommended that the focus shifts from representations to mechanisms, with an emphasis on how close the correspondence between recursive representations and recursive operations is.

Acknowledgements

The first author has been supported by the AGAUR grant 2008FI-00058 and the Anglo-Spanish Society 2008 02 scholarship; both authors have been supported by the Spanish Ministry of Education grant SEJ2006-11955.

References

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

Boeckx, C., & Uriagereka, J. (2007). Minimalism. In G. Ramchand & C. Reiss (Eds.), *The Oxford handbook of linguistic interfaces* (p. 541-574). Oxford, England: Oxford University Press.

Chomsky, N. (1967). Recent contributions to the theory of innate ideas. *Synthese*, *17*, 2-11.

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

Corballis, M. (2007). The uniqueness of human recursive thinking. *American Scientist*, *95*, 240-248.

Di Sciullo, A. M., & Isac, D. (2008). The asymmetry of merge. *Biolinguistics*, *2*, 260-290.

Epstein, S. D., & Hornstein, R. (2000). *Working minimalism*. Cambridge, MA.: The MIT Press.

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

Hinzen, W. (2008). Prospects for an explanatory theory of semantics. *Biolinguistics*, *2*, 348-363.

Maratz, A., Miyashita, Y., & O'neil, W. (2000). *Image, language, brain*. Cambridge, MA.: The MIT Press.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. San Francisco: W. H. Freeman & Company.

Medeiros, D. (2008). Optimal growth in phrase structure. *Biolinguistics*, *2*, 152-195.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behaviour*. New York, New York: Holt, Rinehart and Winston, Inc.

Miller, P. H., Kessel, F. S., & Flavell, J. H. (1970). Thinking about people thinking about people thinking about...:a study of social cognitive development. *Child Development*, *41*, 613-623.

Neeleman, A., & Van de Koot, J. (2006). On syntactic and phonological representations. *Lingua*, *116*, 1524-1552.

Oppenheimer, L. (1986). Development of recursive thinking. *International Journal of Behavioural Development*, *9*, 401-411.

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

Pylyshyn, Z. (1984). *Computation and cognition*. Cambridge, MA.: The MIT Press.

Pylyshyn, Z. (1988). Computing in cognitive science. In M. I. Posner (Ed.), *Foundations of cognitive science* (p. 49-92). Cambridge, MA.: The MIT Press.

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

Samuels, R. (forthcoming). Classical computationalism and the many problems of cognitive relevance. *Studies in History and Philosophy of Science*.

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

Soschen, A. (2008). On the nature of syntax. *Biolinguistics*, *2*, 196-224.

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

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