Recursion Isn't Necessary for Human Language Processing: NEAR (Non-iterative Explicit Alternatives Rule) Grammars are Superior

Article *in* Minds and Machines · January 2014

DOI: 10.1007/s11023-014-9341-y

CITATIONS
2

READS
595

2 authors:

Kenneth R. Paap
San Francisco State University
86 PUBLICATIONS **4,075** CITATIONS

SEE PROFILE

Derek Partridge
University of Exeter
137 PUBLICATIONS **1,648** CITATIONS

SEE PROFILE

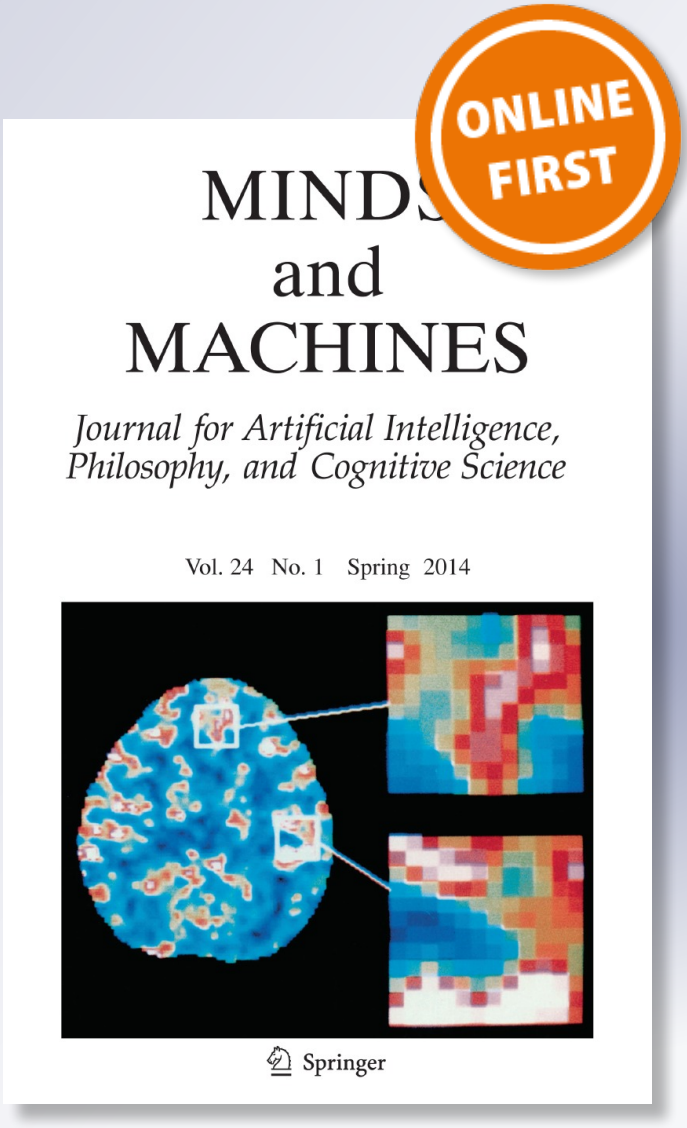
Recursion Isn't Necessary for Human Language Processing: NEAR (Non-iterative Explicit Alternatives Rule) Grammars are Superior

Kenneth R. Paap & Derek Partridge

Minds and Machines
Journal for Artificial Intelligence,
Philosophy and Cognitive Science

ISSN 0924-6495

Minds & Machines
DOI 10.1007/s11023-014-9341-y



Springer

Minds & Machines
DOI 10.1007/s11023-014-9341-y

Recursion Isn't Necessary for Human Language Processing: NEAR (Non-iterative Explicit Alternatives Rule) Grammars are Superior

Kenneth R. Paap · Derek Partridge

Received: 1 August 2013 / Accepted: 16 January 2014

Abstract Language sciences have long maintained a close and supposedly necessary coupling between the infinite productivity of the human language faculty and recursive grammars. Because of the formal equivalence between recursion and non-recursive iteration; recursion, in the technical sense, is never a necessary component of a generative grammar. Contrary to some assertions this equivalence extends to both center-embedded relative clauses and hierarchical parse trees. Inspection of language usage suggests that recursive rule components in fact contribute very little, and likely nothing significant, to linguistic creativity. Further than this, if the productivity of human language is considered as not rigidly bound, but not infinite, then the need for any sort of iteration in generative grammars vanishes and can be replaced with a Non-iterative Explicit Alternatives Rule grammar. The knock-on effects of dispensing with recursive (or any iterative) grammar components are: that language diversity can simply be based on rule and lexicon combinatorics with no potentially infinite dimensions derived from recursive or iterative components of rules; the oddity of a vast, multiply-infinite competence set of 'grammatical but unacceptable' productions is gone; and the development of a language faculty based on rules that eschew iterative rule components avoids any need for explaining 'special' mechanisms. On the broader front of searching for the mechanisms of mind, our analysis can be similarly applied to the proposals for a recursive basis for mind as an explanation for humanity's great leap forward.

Keywords Recursion · Human language faculty · Language processing

K. R. Paap (✉)
Department of Psychology, San Francisco State University, San Francisco, CA 94132, USA
e-mail: kenp@sfsu.edu

D. Partridge
Department of Computer Science, University of Exeter, Exeter, UK

Published online: 28 January 2014

Concisely defined, recursion is a self-referential process. In the more specific framework of a context free grammar (CFG) a generic example of a recursive rule would be: S → aSb where a nonterminal on the left can be rewritten to include another token of itself on the right.

References to recursion with respect to the human language faculty recur throughout the language science literature and have done so ever since Chomsky (1956, 1965) proposed this "solution" to the requirement of accounting for infinite linguistic capacity based on a finite means. However, we take the 2002 paper by Hauser, Chomsky and Fitch as the modern basis for consideration of recursion with respect to human language: it is presumably consistent with Chomsky's current thinking on this issue; it makes fundamental claims about recursion and language; and it has triggered a wide variety of responses to its central theses. Hauser et al. draw a distinction between the faculty of language in the broad sense (FLB) and in the narrow sense (FLN). FLB includes a sensory-motor system, a conceptual-intentional system, and the computational mechanisms for recursion, providing the capacity to generate an infinite range of expressions from a finite set of elements.

Our interests focus on the kinds of claims that linguists and cognitive scientists have attributed to this article and the impact of our analysis on those claims and for the mechanisms that are required to account for or explain FLN. Van der Hulst (2010) lists five bold and influential claims that were sparked by the Hauser et al. article:

1. Recursion essentially constitutes the innate human language faculty
2. Recursion is the sole uniquely human trait of human language
3. Recursion is unique to the language faculty
4. Recursion is universal (i.e. present in all human languages)
5. Recursion is unique to the human mind

Like van der Hulst we acknowledge that there is uncertainty regarding what Hauser et al. precisely claim and that different interpretations can be found in the many responses to the article. The argument developed in this article leads to the conclusion that recursion is neither a necessary nor preferred modeling choice for human language production or understanding. If recursion is abandoned and replaced with better and more realistic options then claims (1) through (4) can no longer be true, and consequently claim (5) needs reassessing. We organize our account by examining seven common misconceptions about the relationship between recursion and language.

Myth 1: There are "Recursive" Structures

Inferences from Informal Definitions

In many of the relevant papers, we find it difficult to pin down important details with respect to the claims made about recursion. Firstly, do the authors mean "recursion"

in the purely technical sense, i.e., a self-referential mechanism? Or do they use the word more loosely to refer to any repetitive mechanism, such as can be found in an iterative looping mechanism that need not be self-referential in the technical sense? Secondly, is this term, however interpreted, proposed as merely a notational device to specify a grammar with no implications with respect to the processes that occur in the FLN? Or is there a commitment to recursive mechanisms as a necessary cognitive component to account for the infinite novelty of human language? In summary, we must distinguish between recursion as a self-referential device and other iteration devices (i.e., loops) more generally, and between its use for notational convenience and its use as a cognitive mechanism. Throughout our discussion we assume that referenced authors do mean 'recursion' in the technical sense of a self-referential mechanism.

The Illusion of "Recursive Structures"

The occurrence of structures that could have been produced by a recursive process does not imply the need for a recursive generator. Language scientists sometimes use the noun phrase "recursive structures" in contexts that imply that recursion can be defined as either a process or a structure. For example, Mithun (2010), observes that "A definition of recursion that is consistent with much current work in linguistics is that of Pinker and Jackendoff (2005) whereby a recursive structure is characterized as 'a constituent that contains a constituent of the same kind'" (p. 17, emphasis added).[1] References to "recursive structures" would be somewhat misleading, but not wrong, if recursive structure was simply a shorthand for the more cumbersome but correct statement that many sentences have structures consistent with the application of recursive rules. Tiede and Stoute (2010) have also called for the same distinction: "… recursive structures are distinct from recursion which is a process" (p. 151). A more precise and constrained aspect of this misconception is that there are certain types of sentences that can only be produced by recursive processing. The case of center-embedded sentences is considered next as a first example. For this case and other cases we will show that any sentence that can be produced by recursion can also be produced by looping grammars or (within empirically defensible limits) a small set of non-iterative specific rules.

Myth 2: Only Recursion Can Generate Center-Embedded Sentences

Christiansen and Chater (1999), Corballis (2007), Harder (2010), Kinsella (2010) and Pinker and Jackendoff (2005) have all acknowledged that the process of tail

[1] Mithun quotes only the latter part of the full definition offered by Pinker and Jackendoff: "Recursion refers to a procedure that calls itself, or to a constituent that contains a constituent of the same kind)" (p. 203). Although Pinker and Jackendoff do not themselves use the term "recursive structure" it appears that Mithun interprets the "or" clause as meaning that another way to view recursion is in terms of a specific type of constituent structure.

recursion can be mimicked by iteration, but have all conjectured or concluded that the type of center-embedded recursion illustrated in Sentence (1) cannot. Tail recursion itself can be either left-branching as in Sentence (2) or right-branching as in Sentence (3).

1. The cat that the dog that John saw chased bit the mouse.
2. The big fat black dog was sleeping.
3. John saw the dog that chased the cat that bit the mouse.

Any grammatical sentence produced by recursion can also be produced by loop iteration; these are two formally equivalent options for producing or parsing languages.[2] Sentence (1) with two levels of center embedding will be used as a prototype to demonstrate that a grammar that uses loop iteration rather than recursion can both generate and recognize center-embedded sentences.

The upcoming examples will unfold within the framework that a grammar defines a set of strings. Unless the grammar is a simple listing of the set, then it must include some defined process to do the transformations from a rule set to a string set. Thus, the formal definition of a grammar must define both the permissible rule structures and how they operate. A bare bones CFG is so deceptively simple that, in practice, the two parts of the definition are rarely made explicit. Informally they can be stated this way: (1) A permissible rule has a non-terminal expression on the left side and either a string of terminals and/or non-terminals on the right side. (2) The only valid operation is that the right side of the string can be substituted (rewritten) for all occurrences of the left side non-terminal string and this re-writing repeats until all non-terminals have been eliminated. This results in a grammatical string in the language. In summary, the bare bones process defined in a standard CFG is simply one of rewriting one string into another until the string is all terminal symbols, in which case it is an element of the language set and, of course, a simple reversal of the rewrite process determines the language recognized by the grammar. By way of a compact example, the following set of CFG rules:

$$\mathrm{S} \rightarrow \mathrm{A}b\ c$$
$$\mathrm{A} \rightarrow e\ f$$

produces a grammatical string set consisting of $e\ f\ b\ c$ because this is the only rewrite possible.

One might argue that looping processes are not allowed in a bare bones CFG because there is no provision for loop structures and the processing they define. Looping processes are, of course, standard fare in programming languages and there

[2] It is generally believed by computer scientists (although perhaps not indisputably proven) that in programming there is an absolute equivalence between recursion and loop iteration, i.e., what can be specified recursively can alternatively be specified iteratively, and vice versa. In human language the claimed recursive structures are limited. The complex, multi-stack, mutually recursive structures found in software have not been postulated for language. Consequently, a claim of equivalence is sound in the language domain: any recursively formulated grammar rule can be rewritten as an iterative equivalent.

appears to be no reason in principle to block their inclusion in formal grammars. In contrast to the apparent novelty of rules governing loop iteration, recursive rules easily slide into the standard CFG framework because, on the surface, they are simply self-referential rewrite rules with perhaps the oddity of a 'nothing' option to halt the recursive looping:

$$S \rightarrow a\ B\ c$$
$$B \rightarrow e\ B\ f \mid {}'\text{nothing}'$$

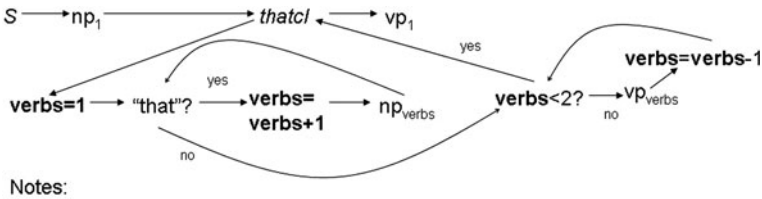
Here are the two rules for a small iterative (but non-recursive) grammar that can generate and recognize center embedded relative clauses all the way to infinity:

Iterative Grammar 1
Rule 1: S → np_1 thatcl vp_1
Rule 2: thatcl → **verbs** = 1, **while** "that" (add 1 to **verbs**, np_{verbs}), **for** i = **verbs** **downto** 2 **do** vp_i

The processing assumptions for this grammar, which follows standard while-loop and for-loop definitions, are illustrated in Fig. 1. We will show first how this iterative grammar can recognize that an expression is a proper member of the string set for this language and then, in a similar manner, show how it could generate the same sentence.

Recognition

When confronted with a potential sentence, S, the first rule expects to match an initial noun phrase with the syntactic component np_1. After matching np_1 there is a reference to the thatcl rule. This rule sets a local variable **verbs** to have the value 1



Notes:

thatcl rule composed of three components

1. set counter variable **verbs** with initial value 1
2. a **while** loop that is traversed every time "that" is found (during recognition), or is produced (during generation), if no instances of "that" found/produced then **while** loop will be totally skipped. otherwise the np is found/generated on every iteration is associated with current value of **verbs**
3. a **for** loop; number of times traversed controlled by value of **verbs** on entry (it iterates (**verbs**-1) times, which will be zero if no np found/generated in **while** loop), on every iteration a vp is found/produced and associated with current value of **verbs**

Fig. 1 Graphical specification of the processing associated with Iterative Grammar 1

and then expects to find "that" as the next word, if successful it enters the **while** loop, if not it jumps the **while** loop to process the next (and final) component of this rule, the **for** loop. This loop iterates from i = verbs, which is 1 in this case (because **while** loop was jumped because no "that" was found) down to the final value i = 2, but as 2 is greater than 1, there is no iteration to be done (no vps to expect), and so the **for** loop is finished, although it's recognized nothing. Then rule 2 is also finished and control returns to rule 1 where a final vp_1 is expected.

When confronted with a potential sentence, which does have at least one "that" clause, say Sentence (1), the first rule expects to match an initial noun phrase with the syntactic component np_1 yielding $[\text{The cat}]_1$. Next is a reference to the thatcl rule. This rules sets a local variable **verbs** to have the value 1 and then expects to find "that" as the next word and if successful it enters the **while** loop. Within this loop the grammar first increases the value of **verbs** by 1, then it expects to find an np which it indexes with the value of **verbs**, i.e., the first np found inside the **while** loop is $[\text{the dog}]_2$. Control returns to the front of the **while** loop and another "that" is found. Then **verbs** is incremented to 3 and associated with the next np yielding $[\text{John}]_3$. When the **while** loop is retested there are no more "that"s so the while loop processing terminates and the next rule component, the **for** loop, is entered. In this case, the initial value of i is 3, which is greater than the termination value, 2, so the body of the **for** loop is entered and the expected vp is recognized as $[\text{saw}]_3$. (If a vp is not found, then we have a parse failure and the expression is not a grammatical sentence.) Now the **for** loop index i is decreased by 1 and so becomes 2 which is not less than the termination value 2, The **for** loop is re-entered again and this time instantiates $[\text{chased}]_2$ as the expected vp. When **verbs** is decremented to 1 the **for** loop terminates with success and control returns to the S rule where a final vp is expected and instantiated as $[\text{bit the mouse}]_1$. So the sentence "The cat that the dog that John saw chased bit the mouse." is recognized successfully as $[\text{The cat}]_1[\text{that}][\text{the dog}]_2$ [that] $[\text{John}]_3$ $[\text{saw}]_3$ $[\text{chased}]_2$ $[\text{bit the mouse}]_1$ in that order.

In principle it should be clear that the recognition of center-embedded that clauses can go on to infinity. As the **while** loop recognizes a succession of "that" +np fragments the value of **verbs** increases (and labels the nps), and whenever no more "that"s are found the **while** loop terminates and recognition moves on to the for loop which (controlled by the value of **verbs**) is configured to recognize exactly the same number of vps and to associate the correct label with each by counting down from the final value of verbs, set by the **while** loop, to the value 2.

Generation

Now consider the generation of Sentence (1): the S rule generates the np "The cat" and associates it with 1 resulting in $[\text{The cat}]_1$. The generation process encounters the reference to the thatcl rule: this rule sets **verbs** = 1, and may or may not generate the word [that]. Because we're assuming that in this case the intention is to express the propositions contained in Sentence (1) "that" will be generated and the

while loop entered: so **verbs** is increased to 2 and we have an np generated (e.g., $[\text{the dog}]_2$), and the process returns to the front of the **while** loop. This re-entry also generates "that", increases the value of **verbs** to 3, and another np is generated (e.g., $[\text{John}]_3$). When the beginning of the **while** loop is entered this time we assume that another "that" is not generated and control moves along to the next (and final) component of the thatcl rule—the **for** loop. This loop is entered only if value of **verbs** $\geq$2, it is 3, so the **for** loop generates a vp and associates it with the value 3, (e.g., $[\text{saw}]_3$), the value of **verbs** is decreased by 1 and loop entry condition retested. Because the value of **verbs**, 2, is greater than or equal to 2 the loop is re-entered. Another vp is generated, (e.g., $[\text{chased}]_2$, the value of **verbs** decreases to 1, and the **for** loop entry condition fails as **verbs** $= 1$ is not greater than or equal to 2. Rule 2 has terminated successfully and control returns to rule 1 at the point following the reference to rule thatcl. The final step of rule 1 is to generate vp_1, (e.g., $[\text{bit the mouse}]_1$). Thus, the final string generated is $[\text{The cat}]_1[\text{that}][\text{the dog}]_2$ [that] $[\text{John}]_3$ $[\text{saw}]_3$ $[\text{chased}]_2$ $[\text{bit the mouse}]_1$. Not only is it a grammatically acceptable string of embedded that clauses, but it also contains the essential information explicitly specifying the np-vp relationships.

Myth 3: Only Recursion Can Build Hierarchical Structures

Pinker and Jackendoff (2005) took the distinction between recursion and loop iteration a step further by asserting that only recursion "creates hierarchic structures governing the linear strings it produces, typically by repeatedly subordinating an element to another instance of the same type … Tail recursion cannot be mimicked by iteration when it comes to computations that require more than duplicating input–output behavior ('strong generative capacity')… such as inferences that depend on the grouping and labeling of constituents" (p. 211). Likewise Karlsson (2010) asserts: "Their main difference is that recursion builds structure by increasing embedding depth whereas iteration yields flat output structures which do not increase depth…" (p. 43).

This conjecture is wrong within the following framework. Bare bones CFGs, even recursive ones, do not build representations beyond strings, linear sequences of the terminal symbols. Additional processing assumptions (such as, syntactic significance for grammar non-terminals) need to be made in order to build parse trees or other syntactic structures that may serve subsequent processes leading to sentence comprehension. We will first show how the parse tree shown in Fig. 3 can be built from a recursive grammar (Fig. 2), but the main goal is to demonstrate that the same tree can be constructed from an iterative looping grammar. It is the parsing performed by an algorithm (whose only constraint is that it must recognize/generate the same language as the grammar defines) not the grammar itself, that determines the structures or representations available for subsequent stages of comprehension. This parsing must include processes beyond those that define a CFG if the resultant structure is to be more than a string of terminals.

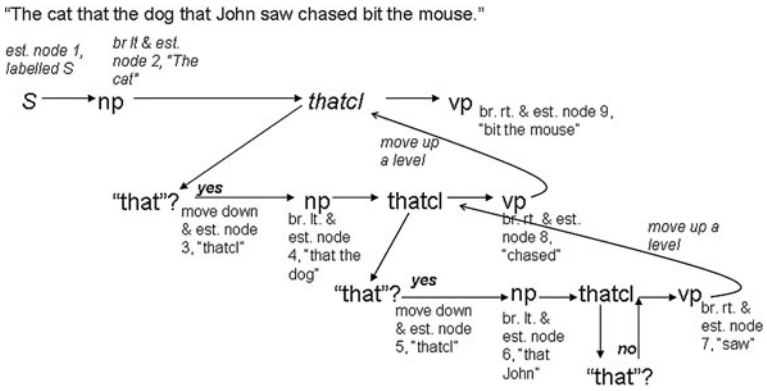


Fig. 2 A simple recursive grammar and a possible parse process for Sentence 1. Note. *est.* establish, *br.* branch, *rt.* right, *lt.* left

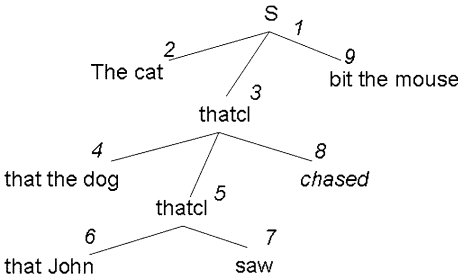


Fig. 3 The parse-tree produced by the parsing algorithm for all three grammars. The *numbers* indicate the order in which both processes establish the nodes

Recursive Grammar 1
Rule 1: S → np thatcl vp
Rule 2: thatcl → "that" np thatcl vp l'nothing'

In summary, this parsing algorithm initiates a new level (by means of a 'trunk' node labeled "thatcl") in the tree-structured hierarchy on each recursive re-entry of the thatcl rule ("move down" in figure) and returns to the higher level trunk node on each exit from this rule ("move up" in figure). A right branch is constructed whenever an np is encountered and a left branch whenever a vp is found, in each case control returns to the trunk node of the current level after building the branch.

The recursive parse schematically shown in Fig. 2 provides a pictorial representation of how an actual parsing algorithm might construct the parse-tree shown in Fig. 3. To counter any concerns that we are hand waving, a parsing algorithm is presented next for the Recursive grammar shown in Fig. 2 and later for both the Iterative and NEAR grammars. This exercise demonstrates that the same hierarchical parse-tree for Sentence 1 can be built from explicit parse algorithms that use recursion, looping, and alternative rules; respectively.

Recursive Grammar Parsing Algorithm

(This procedure builds a left branch and labels the new terminal node with 'prefix np'.)

```
Npbuild(prefix,np) procedure is
  Branch left & attach "prefix" np
```

(This procedure builds a right branch and labels the new terminal node with 'vp'.)

```
Vpbuild(vp) procedure is
  Branch right & attach vp
```

(This recursive procedure checks for "that", if found it builds down recursively, constructing left branches until no more "that"s encountered, and then branching right as the recursion unwinds. It echoes Rule 2)

```
Thatclbuild procedure is
 If "that" then
          move down a level, establish node and attach "thatcl"
          Npbuild(that,np)
         Thatclbuild
          Vpbuild(vp)
          move up a level
       else return
```

(This is the main procedure; it echoes Rule 1 and builds the complete parse tree.)

```
Streebuild procedure is
  set tree root node and attach label "S"
  Npbuild( ,np)
 Thatclbuild
 Vpbuild(vp)
```

For those who are not hardened recursive programmers we include a narrative in "Appendix" that describes in detail how this recursive process builds the parse tree. The iterative grammar used above to show that loop iteration can both recognize and generate center embedded sentences would need a similar augmentation in order to build the parse tree illustrated in Fig. 3. As for the recursive parse, our parsing algorithm echoes the structure of the grammar upon which it is based.

In our complete parsing algorithm based on the looping grammar, we have chosen to collapse the two looping grammar rules into one parsing procedure.

We could have two procedures each more directly reflecting a grammar rule. Alternatively, we could have a single-rule, equivalent looping grammar, such as:

S → np_1, **verbs** = 1, **while** "that" (add 1 to **verbs**, np_{verbs}), **for** i = **verbs** **downto** 2 **do** vp_i, vp_1

which is directly reflected in the parse process specified below.

Notice that this particular choice is not an option when recursion is the vehicle of choice because recursion demands a non-terminal in the grammar to be self-referenced, and similarly it requires a separate procedure in the parse process. Furthermore, sometimes a certain rule ordering is required to ensure that the recursive "stopping condition" functions correctly. In contrast, a simple CFG has no restrictions on rule order. Notice also that the non-terminal "thatcl" does not occur in the equivalent single-rule grammar which further reinforces the point that useful parse trees are constrained by their grammars, but determined by the parsing algorithm.

Looping Grammar Parsing Algorithm

(this single procedure parses and builds the desired parse tree)

Streebuildit(sentence)

set parse tree root and attach "S"

match np, branch left and attach np to node

verbs=1

(this loop builds down and adds left branches as successive "that"+np sentence fragments encountered)

while "that" found

move down, establish trunk node and attach "thatcl"

verbs=verbs+1

match next np, branch left, establish node and attach "that" np

(this loop adds right branches and moves up as successive vps encountered)

for i=**verbs** downto 2

match vp, branch right, establish node and attach vp

move up a level

(this statement adds final vp)

match vp, branch right, establish node and attach vp

For sentence (1) this parse algorithm would be successful and the parse tree built has the same structure, built in the same order, as the recursive parse. The combination of a 'matched' while loop and for loop, in effect duplicates the hidden functionality of a stack.

The looping-grammar parse appears to require more 'baggage' than the recursive-grammar—the local variable **verbs** and subscripted nps and vps. However, the stacking that is necessary for the recursive-grammar parse will involve making and storing, in order, copies of all information local to procedure Thatclbuild. Stack space is not just stacking the succession of nps but each recursive re-entry of procedure Thatclbuild requires saving a complete copy of all information needed to later restart the procedure in exactly the same context as it was halted. None of this overhead is explicit in the description of the parsing algorithm, but it is arguably more space and time consuming than the looping-grammar algorithm in which the overheads are all on show.

It is worthwhile to elaborate that a recursion mechanism requires sophisticated discontinuous processing—repetitions of: halt current processing, save all the current processing information, restart processing from last halt by re-establishing saved information. In contrast loop iteration does not require the same degree of discontinuous processing. It is a 'smoother' process with only one-time stopping, saving and restarting as is demanded by the basic re-write rule function (and not even that in the case of our single-rule grammar). Recursion involves a demanding requirement for precise and complex discontinuous processing. Loop iteration is more linear. This difference may be pertinent to the development of theories of cognitive processes for language processing.

Myth 4: Recursion is Needed to Account for Infinitely Long Sentences

The following linkage is often implied: (1) languages can be viewed as a set of sentences, (2) the set is infinite, (3) discrete infinity is assured because a single grammatical sentence can be of unlimited length, and (4) this compels the need for a recursive grammar. Pullum and Scholz (2010) call the assumption of discrete infinity the "infinitude claim" and critique in detail many assertions (by Epstein and Hornstein 2004; Hauser et al. 2002; Huddleston 1976; Langacker 1973; Lasnik 2000; Stabler 1999; Yang 2006) that the infinitude of sentences is an established linguistic universal that characterizes every human language. Given the prominent role that Hauser et al. have played it is instructive to look at their framing of the infinitude claim: "All approaches agree that a core property of FLN is recursion, … FLN takes a finite set of elements and yields a potentially infinite array of discrete expressions. This capacity of FLN yields discrete infinity (a property that also characterizes the natural numbers)" (p. 1571).[3] Van der Hulst sums it up this way: "…linguists generally assume that the

[3] This common analogy contains a crucial weakness: the natural numbers contain only one dimension of infinity. Hence they can be strictly ordered with respect to size. Natural languages are not limited in this way (they contain, for example, a dimension of length infinity and one of structural diversity) and hence sentences cannot be simply ordered. It is precisely this difference that is being overlooked.

infinitude claim is true. And therefore, these linguists design grammars that have recursive mechanisms" (p. xxii).

Strictly speaking, a set or a language (when viewed as a set of sentences) cannot be said to be recursive.[4] In a way similar to the point we made earlier about "recursive structures" this usage can only be shorthand for something like a "recursively enumerable set" or a "recursively generated language." Recursion and iteration are mechanisms by which sets may be generated or recognized. The structure of a given set may suggest a neat recursive generative mechanism, but that is merely a suggestion of one way that the set may have been produced (or could be analyzed). When restricted to finite means, infinite sets can only be produced by recursive or looping mechanisms. But because there is a formal equivalence between these two options—what can be specified recursively can also be specified as loop iteration, and vice versa—nothing within the structure of an infinite set can determine the choice. Other criteria, such as elegance or efficiency, might be used to tip the balance one way or another, but the structure of the set of elements by themselves can have nothing to say on this point. In summary, if one accepts the commonly held assumption that a single grammatical sentence can be of unlimited length then one must propose grammars that include either recursion or loop iteration, but the characteristics of the set itself cannot adjudicate between the two formally equivalent options.

Myth 5: Infinitely Long Sentences Capture Human Language Diversity

Recursive grammars generate competence sets that are infinite in many dimensions, one for each recursive rule component. Consider Fig. 4, that shows some novel variations on a familiar sentence. These variants constitute no more than a very small portion of the endless human capability for novelty and creativity. However, this is not an undifferentiated mass. It varies along a number of different dimensions and two significant ones can be pulled apart. The four arrows, each heading off to its own infinity, loosely illustrate the directions in which four different recursive grammar rules will create endless sentence variations. Thus the top-right arrow follows the grammatical productions of our old friend the <that clause> embedding rule. The top-left arrow charts the endless possibilities for adjectives to describe the "fox". The bottom-left arrow marks the path of endless qualification of the verb "jumped", and finally, the bottom-right arrow indicates the endless possibilities for adding a new sentence on to an old one by inserting "and". Clustered loosely around the central "seed" sentence are all sorts of more creative modifications. They result from all the word and phrase substitutions, additions or rearrangements that a huge vocabulary and a complex grammar make possible. Along each of the four arrows something different is happening in detail, but all four can be viewed as treading the same path to infinity—one of continual growth in sentence length. In

[4] This stricture does not apply to the so-called formal languages, such as those use to define grammars. The potential for confusion stems from the use of the word 'language' for two very different classes of phenomena: formal languages which may include all sorts of extra apparatus, such as explicit looping constructs; and natural languages which are primarily a sets of strings of words (or perhaps sounds).

Fig. 4 Four possible pathways for expanding a familiar sentence to discrete infinity

general, recursive grammar rules define an infinite sequence of longer and longer sentences. The language creativity they provide is founded on unfaltering expansion of sentence length.

Ever-increasing sentence length is a strategy that plays no part in human creativity with language. Rather we delete, insert or substitute words and phrases, and re-arrange their orderings. Furthermore, we do so in service of conveying intended thoughts. Language creativity is founded on re-structuring not inexorable growth in sentence length. The human dimension can usefully be called *structural diversity* distinct from the longer and longer sentences of recursion-generated diversity: *length diversity*.

In Fig. 5, which is meant to be three-dimensional, length diversity is shown in the plane of the page, and structural diversity is orthogonal to this plane. To be more specific, the structural diversity within the central tube is the result of modifications needed to convey a vast variety of intended thoughts. Such "fiddling" with sentences by selecting one word here but another one there may, mistakenly, seem minor compared to the elegant flights to infinity guaranteed by recursive grammar rules. Perhaps this is why such rules have generally been favoured by language scientists devising a basis for the boundless creativity of human language. In summary, iterative grammar rules (using either recursion or looping) account for a diversity of language that humans virtually ignore. It is the wrong sort of diversity.

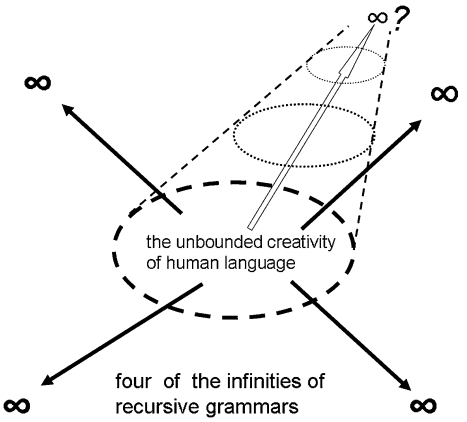


Fig. 5 A schematic of our argument that human language creativity lies within a "tube" of structural diversity that is unrelated to the pathways to infinity generated by, in this example, four different recursive rules. It is within this "tube" of structural diversity that human language creativity lies. As you can see recursive grammar rules shoot straight out of it as soon as a few (of the potentially infinite number) of recursive repetitions have been exploited

As illustrated next, structural diversity is not just another variant of length diversity, it is an orthogonal dimension and the one that reflects actual language creativity.

The right sort of diversity reveals itself in an examination of actual language use that overwhelmingly prefers structural diversity to length infinity. By structural diversity we are referring to the fact that there are an extremely large number of novel sentences each of finite length. In practice it can be impossible to distinguish between a potentially finite set that exceeds our necessarily finite resources to explore it, and a truly infinite set. In the former case, the set can in principle be generated from a finite set of rules without recourse to either recursion or looping. The vast, but finite, number of set elements is generated from the combinatorial interaction of the rule components. In the case of human language, the grammar rules combine not only with each other but also with a vast lexicon of, say, 50,000 words that is unbounded and continuously expanding.

If the lexicon is viewed as fixed for a particular individual and point in time and the rules do not allow recursion or loop iteration, then the set of potential sentences is finite. Would this set be adequate to account for the structural diversity of language, that is, for the capacity to produce (or recognize) the full range of novel sentences? Pullum and Scholz (2010) believe that the "… infinitude of the set of all grammatical expressions is neither necessary nor sufficient to account for creativity of language." (p. 126). Harder (2010) agrees that: "Recursion may not be the right way to account for linguistic creativity…" (p. 234). Part of the argument presented by Pullum and Scholz is built on the example of Haiku, the highly structured form of Japanese poetry. Although accomplished Haiku poets are revered for their creativity their outputs are "selected" from a finite set of 10^{34} conforming utterances. To put this number in perspective physicists estimate that less than 10^{25} microseconds have elapsed since the big bang (Harel 2000).

Humans are also highly flexible in their spontaneous use of language, particularly in their ability to respond verbally to novel circumstances or to reformulate previously expressed propositions in new ways. We agree with Pullun and Scholz in their conclusion that abandoning the infinitude claim in favor of an extremely large finite set of sentences would not imperil the claimed semantic universal that all human languages have "effability" (Katz 1978), that is, the property of having resources to express any proposition. Perhaps one caveat to this conclusion would be the need to expand the finite set through the addition of new lexical items to accommodate some types of new knowledge.

Myth 6: Performance Intrudes Significantly into Infinite Embeddings

Is the incidence of "recursive structures" in natural language frequent enough and deep enough to justify recursive grammars? No, a grammar that enables length infinity via either recursion or loop iteration is promiscuously over productive. The empirical facts are that humans do not understand (nor casually can generate) sentences with more than about two or three levels of embedding. Karlsson's (2010) analyses show: "…that multiple nested syntactic recursion of degrees greater than 3 does not exist in written language, neither in sentences nor in noun phrases or prepositional phrases. In practice, even nesting of degree 2 is extremely rare in writing. In speech, nested recursion at depths greater than 1 is practically non-existing. Left-branching tail-recursion of clauses is strictly constrained to maximally two recursive cycles. Right-branching clausal tail-recursion rarely transcends three cycles in spoken language and five in written language. On constituent level both left- and right-branching is less constrained (especially in written language), but left-branching genitives rarely recurse more than two cycles" (p. 43).

On the basis of an examination of other language corpora Laury and Ono (2010) similarly conclude that the reality of recursion "… is being questioned for the first time with overwhelming evidence that it has little to do with what actual speakers know and do" (p. 85). We agree with van der Hulst (2010) that formally representing certain linguistic phenomena with recursive devices is a modeling choice and that other modeling choices should be considered. Evaluating these alternatives is difficult because, as van der Hulst puts it, "… there is a dispute on what the linguistic phenomena are when one does not wish to rely on 'grammatical intuitions' which, supposedly, reflect the principles of competence without being cluttered by performance factors." (p. L). While acknowledging the matter is unsettled, we side with those who adopt a more empirical stance, and find value in inspecting what occurs in actual language use. As described above, the types of constructions that can be modeled in terms of recursive devices are not that common, and, when present, are of severely limited depth.

In a comprehensive summary of the relevant literature in psycholinguistics Christiansen and MacDonald (2009) suggest that the reason why center-embedded sentences are rare is because they are difficult in a variety of ways. They are hard to

repeat with proper intonation, to recall, to paraphrase, to correctly understand and, furthermore, their understanding is quite resistant to training. The difficulty has traditionally been attributed to extrinsic memory limitations. The variants, as summarized by Christiansen and MacDonald: "… include limits on stack depth (Marcus 1980), limits on the number of allowed sentence nodes (Kimball 1973) or partially complete sentence nodes in a given sentence (Stabler 1994), limits on the amount of activation available for storing intermediate processing products as well as executing production rules (Just and Carpenter 1992), the "self-embedding interference constraint" (Gibson and Thomas 1999), and an upper limit on sentential memory cost (Gibson 1998)" (p. 129). As connectionists Christiansen and MacDonald offer a more unified view that the limitations are due to interactions between linguistic experience and architectural constraints on learning and processing. The important point for our argument is simply that center-embedded sentences are limited in practice to only one or two levels of embedding.

The severe limitation on the depth of embedding in actual performance gives rise to the notion of "grammatical but unacceptable" sentences. The infinite competence set contains a variety of differently classed utterances that in total constitute a native speaker's knowledge of, say, the English language. All will be, by definition, 'grammatical', some will also be 'acceptable' with a variety of possible further qualifications on 'acceptability', such as perhaps 'spoken' rather than 'written' as well as cultural and geographical qualifiers. The infinite set will also contain the 'grammatical but unacceptable' sentences which is those sentences that our knowledge of the language would allow us to understand or generate were it not for limitations of our cognitive architecture. This position compels the adoption of grammars that generate infinitely more unacceptable sentences than acceptable ones, a position that should become increasingly uncomfortable.

Myth 7: Recursive Processes are Simple, Elegant, and Preferred

Let's return to the idea that the redeeming virtue of recursion as a modeling choice is its elegance or simplicity. Tiede and Stoute (2010) in their chapter on the relationship between infinity and recursion opine that: "…the most important, reason for assuming infinity was originally put forward by Chomsky in 1956: 'In general, the assumption that languages are infinite is made for the purpose of simplifying the description. If a grammar has no recursive steps […] it will be prohibitively complex—it will, in fact, turn out to be little better than a list of strings…' (Chomsky 1956: 115–116)" p. 154.

Three aspects of this argument can be challenged. First, the quest for simplicity cannot trump all other considerations. If the simplest model distorts too much, then some degree of complexity should be introduced. For us, the excessive over productivity of recursive grammars requires such a step. Perfors et al. (2010) agree that: "…a grammar with recursive rules, relative to one without, will overgeneralize, since it predicts very long sentences that are never actually used… All else being equal, a grammar that generates sentences not present in the observed language should be dispreferred…." (p. 161).

Second, contrary to the goals of Chomsky's (1995) Minimalist Program, grammars for human languages may turn out to be far from perfect simplicity, yet not "prohibitively complex". Rather than introducing extra complexity, the non-iterative explicit-alternatives rule we present later is arguably simpler than the iterative rules as it doesn't require the hidden complexities of recursive processing nor does it generate the discrete infinity of unperformable sentences. As a precursor to our recommendation that truly recursive rules can be replaced Perfors et al. (2010)[5] observe that: "as long as there are no sentences with more than n embeddings, a grammar with n non-recursive rules would also parse the language" (p. 160). Finally, Chomsky's leap to the position that grammars without recursion are reduced to "little better than a list of strings" is simply extreme and unjustified primarily because of the very limited performance examples that need to be accounted for. Chomsky's concern would gain traction if performance did in fact intrude significantly into the infinite embeddings, but it does not. As reviewed earlier, it barely scrapes the surface of the recursive possibilities.

By relinquishing the requirement for discrete infinity and settling for the (in practice) indistinguishable alternative of a vast and unbounded performance set, the need for either recursive or iterative-loop grammars vanishes and they can be replaced with a grammar that is non-iterative and consists of rules that specify a small set of explicit alternatives. For the center-embedded 'that'-clause case a sufficient Non-iterative Explicit Alternatives Rule (NEAR) could be:

NEAR grammar:
S → np vp | np "that" np vp vp| np "that" np "that" np vp vp vp

This particular grammar can account for nesting up to a depth of two, and yet it is not Chomsky's horror of being "prohibitively complex … little better than a list of strings". The simple list of alternative grammatical structures avoids all the looping complexity explicit in the iterative version and hidden in the recursive one; it also omits (as does the iterative version) the odd grammatical component of a 'nothing' option, which is required in the recursive version to terminate the recursive looping.

As promised, a parse algorithm based on the three alternative structures listed in the NEAR grammar can build, in the same order, the parse-tree shown in Fig. 3. Using the procedures Npbuild(prefix,np) and Vpbuild(vp) from the recursive grammar parse algorithm, the same hierarchical parse structure can be built using the NEAR grammar with just one additional procedure whose structure echoes that of the NEAR grammar rule.

[5] Perfors et al (2010) embrace a Bayesian approach that trades-off simplicity and over generation. Because the evaluation metric for simplicity they use appears to focus only on the number of rules and ignores processing costs, which are an important hidden aspect of recursive rules, they are more reluctant than us to completely jettison recursion.

(This procedure builds a left branch and labels the new terminal node with 'prefix np'.)

Npbuild(prefix,np) procedure is

Branch left & attach "prefix" np

(This procedure builds a right branch and labels the new terminal node with 'vp'.)

Vpbuild(vp) procedure is

Branch right & attach vp

(This is the main procedure; it echoes the NEAR grammar rule and builds the complete parse tree.)

Procedure NEARtreebuild is:

set tree root and attach label "S"

Npbuild(, np)

if not "that" **then** Vpbuild(vp) *{np vp alternative}*

Without Recursion

else move down a level, establish node and attach "thatcl"

Npbuild(that, np)

if not "that **then** Vpbuild(vp)

move up a level

Vpbuild(vp) *{np that np vp vp}*

else move down a level, establish node and attach "thatcl"

Npbuild(that, np)

Vpbuild(vp)

move up a level

Vpbuild(vp)

move up a level

Vpbuild(vp) *(np that np that np vp vp vp}*

Returning to Sentence (1): *The cat that the dog that John saw chased bit the mouse*.

We begin with a tree root labelled "S", procedure Npbuild branches left and attaches "*The cat*" to the new node. The next word "*that*" triggers the **else** branch which causes branching down a level to a new node labelled "thatcl". At this second level Npbuild(that, "the dog") branches left and labels the new node "that the dog". The next word "that" again triggers the else branch which causes branching down another level to a new node labelled "thatcl". At this third level Npbuild(that, "John") branches left and labels the new node "that John". Vpbuild("saw") branches right and labels the new node "saw". Control moves up a level to level two. Vpbuild("chased") branches right and labels the new node "chased". Control moves up to top level. Vpbuild("bit the mouse") branches right and labels the new node "bit the mouse". The procedure NEARtreebuild is finished and the sentence string has been used up, so the parse was successful.

Now that we have demonstrated that the same hierarchical parse-tree (Fig. 3) can also be built for the NEAR grammar it is worthwhile to revisit the mistaken assumptions that recursive grammars automatically generate hierarchical parse trees that serve subsequent semantic processing and that non-recursive grammars must yield flat structures. As we have demonstrated neither of these suppositions are true. A useful analogy is that grammars are like instructions for stringing beads indigenous to a specific culture. The rules themselves constrain how the necklace can be constructed (or taken apart). For example, some rules specify the order of beads for a specific type of substring. Other rules specify the order of substrings. Following the rules leads to an acceptable one-dimensional necklace, but obviously no concomitant hierarchical structure is built, even if the grammar includes a recursive rule. A creative jeweler may want to create bead-tree pendants (hierarchical structures, rather than just strings) that are consistent with the same rules. The necklace (grammar) rules will obviously constrain the permissible operations, but the artist still has considerable flexibility in terms of the shape and order of the bead tree. Within those constraints he can build the tree top-down, bottom-up, or inside-out. If the artist wants an apprentice to build the same bead tree in the same order he will need to write a beading (parsing) algorithm for the apprentice to follow. To complete our analogy one must consider the fanciful circumstance that there are two neighboring cultures that produce identical necklaces, but that follow different rules: recursive, iterative, or explicit alternatives. As we have shown for the case of center-embedded sentences with two levels of nesting, a parse algorithm can be written for each grammar that produces the same desired hierarchical structure.

Returning to the topic of the relative advantages of the different types of grammar, the NEAR Grammar formulation has much to recommend it: (1) It is amenable to fine adjustment such as empirical study of language performance characteristics suggest (e.g. a nesting depth of three). Whereas a recursive grammar offers only the choice between length infinity or nothing, and looping grammars can be restricted short of infinity but usually at the cost of more loop-control 'baggage.' (2) Its visual simplicity reflects a true processing simplicity. This contrasts with the visual clutter of a looping grammar and the deceptive visual simplicity of recursive grammars; and (3) Closely related to the previous point: the NEAR grammar makes no processing demands beyond the rewrite process of CFGs.

Summarizing the three options—recursion, iterative looping and NEARs:

1. As a grammar specification device:
   - recursion introduces the hidden complexity of self-referential structures and the uncontrollable leap to infinite length which is a (virtually) unused dimension of discrete infinity.
   - iterative looping can be restricted to avoid the unwanted length infinity but requires the extra notational "baggage" of loops, counting, checking bounds, etc.
   - NEARs introduce no extra grammar features and can deliver whatever depth of "reapplication" is desired with simplicity. The additional rule components that enable the generation of acceptable sentences that "look like" recursion are no different in kind from the other rules.
2. As a generative/analytical process:
   - recursion demands, in general, stacking functionality and may demand an order of rule processing.[6]
   - iterative looping necessitates processes for counting, testing and associating count values with grammatical elements.
   - NEARs are standard grammar elements and make no extra demands because the re-write rules fundamental to CFGs are sufficient.

So, on grounds of simplicity of both grammar specification and processing, NEARs is the preferred option, with recursion, which entails the unwanted leap to length infinity, as the worst.

Our argument is similar to and compatible with van der Hulst's (2010) summary of three analyses (Karlsson 2010; Laury and Ono 2010; Verhagen 2010) of language corpora by: "When recursive structures do occur … they may result from specific templatic constructions involving specific lexical items (often derived from clearly non-recursive constructions) rather than abstract recursive mechanisms" (p. xxxiii). With respect to the third analysis van der Hulst concludes "On the basis of actual usage data, Verhagen shows that none of these classic cases actually requires a truly recursive specification. Empirically, a system that uses relatively specific templates is at least indistinguishable from one using general recursion, and is in some respects even more adequate (which has obvious consequences for the issue whether recursion can have been a target of selection) (p. xxxviii)."

Although Verhagen (2010) comes as close as anyone to embracing the full scope of our argument he doesn't join us in urging the total abandonment of recursion in language processing: "By the same token, a general conclusion of the type 'recursion is irrelevant in natural language' is not licensed by the present considerations either. Recursive phenomena exist, and can be an important feature of specific parts of the grammar of a language" (p. 107). Apparently Verhagen

[6] A simple example of a recursive grammar that imposes restrictions on rule ordering is one that defines the natural numbers, 1, 2, 3, …, ∞. r1: <number> is <number> +1; r2: <number> is 1. Must check r1 first, otherwise with potential <number> as 1, r2 generates infinite regress 0, −1, −2, etc.

believes there is no way of dealing with those relatively rare, but legitimate, cases where he thinks "recursive phenomena exist". Perhaps he's missing the distinction between process and product, and that the latter cannot determine the former as we have demonstrated for the "recursive structures" that occur in languages.

Language, and Perhaps Thought, Without Recursive Processing

We have argued that recursion is neither desired nor required to account for normal linguistic creativity. A small set of explicit rules, when combined with a very large lexicon can generate all the grammatical and acceptable sentences of a language. Within their framework, if Hauser et al. were to jettison recursion there would be nothing explicitly left in the basket of FLN and nothing left to explain the uniqueness of human language. However, the framework used by Hauser et al. is not the only perspective on what constitutes the core knowledge of language. Jackendoff and Pinker (2005) in the last part of their commentary present the detailed case that is summarized below.

Jackendoff and Pinker observe that Hauser et al. retain from traditional generative grammar the clear division of labor between a core grammar that is the sole generator of combinatorial structure and a lexicon that is a list of unstructured words and simple morphemes. "HCF identify this core with recursion, in accord with the Minimalist Program, where the rules of grammar are reduced to the basic recursive operation of Merge. Crucially, this core of operations excludes the lexicon, since words are not computational operations, but are rather stored associations of phonological, syntactic, and semantic features" p. 219. Jackendoff and Pinker find this traditional division of labor to be inadequate for explaining a variety of ubiquitous constructions. Among those is this example of a VP construction in which the complement of the VP is not determined by the verb:

(4) He sang/drank/slept/laughed his head off.
(V his head off = 'V excessively')

The underlined complement is not determined by the verb. As Jackendoff and Pinker observe, these constructions preclude the verb taking its own object, e.g. He drank (*scotch) his head off.

On the basis of many such examples Jackendoff and Pinker conclude that human memory must store linguistic expressions of all sizes including idiomatic sentences and that the language-specific part of grammar resides in the nature of these stored representations rather than in the operations that combine them. This approach has considerable empirical support and presents a clear alternative to Hauser et al.'s proposal that FLN consists of the single operation of recursion. In many respects this echoes the views of computational linguists like Schank and Wilks (1974) who have maintained for decades that our language faculty is not primarily syntactic and rule based. Given their goals they were forced immediately to go beyond the simplistic CFGs and to include non-syntactic constraints in order to generate and recognize a realistic set of sentences. To be fair, Jackendoff and Pinker do not dispel recursion from human cognition: "…syntax is the solution to a basic design

problem: semantic relations are recursive and multidimensional but have to be expressed in a linear string. The upshot is that syntax (and hence syntactic recursion) is not to be regarded as the central generative capacity in language, from which all productivity in expression derives. Rather it is a sophisticated accounting system for marking semantic relations so that they may be conveyed phonologically" (p. 223). Although we agree with most of this analysis, it may be premature to conclude that "semantic relations are recursive", an issue that is addressed next.

In his book, *The Recursive Mind,* Corballis' (2011) argues that "the modes of thought that made language possible … were … possessed of recursive properties to which language adapted". He focuses "on two modes of thought that are recursive, and probably distinctively human … [they are] the ability to call past episodes to mind and … the ability to understand what is going on in the minds of others.[7] This too is recursive." He then claims that "most language … is utterly dependent on this capacity" (p. ix). Our analysis denies this latter claim, but also calls into question his bases for labeling the "two modes of thought" as "recursive." For Corballis, "it is in thought rather than in language that recursion originates". So in seeking a definition "that might apply usefully to human thought" (both p. 6) Corballis cites Pinker and Jackendoff (2005) definition (fully quoted in footnote 1) and extends it to include "self-similar embedding." But given that self-similar is an ill-defined concept and that embedding is not a requirement of a technical definition of recursion, his definition must be classed as one of the looser variety. He is thus able to identify "recursion" throughout the human mind but it is no longer recursion in the technical sense.

Although one might want to remain agnostic with respect to the role of recursion in the language of thought because there is no observed set of grammatical and acceptable thoughts, we are not reluctant to revisit one of our fundamental points: examining a set of outputs (whatever they are) cannot adjudicate if the underlying process was based on recursion, loop iteration, or NEARs. Language-of-thought aside, we remain confident that the set of observed grammatical and acceptable sentences do not require the assumption of a recursive cognitive process when recursion is formally defined in terms of self-reference.

Final Discussion

Summary

We pointed out that recursive and iterative processes are equivalent options for grammars to employ in generating a set of strings. More specifically we showed that this is true for the hallmark of "recursive structures", center-embedded relative clauses. We extended this to show that recursion, iteration, and NEARs can build the same parse trees and build them in the same order. These steps show that recursion is not necessary to account for sentences of infinite length nor for complex

[7] This aspect of theory of mind (viz., the ability to understand what is going in the mind of others) conjures recursive-looking structures, "I know that he knows that I know…." that may also have very constrained levels of embedding, even in the best chess or poker players.

parse trees. However, on the basis of actual language use and the extreme over productivity of recursive grammars we argued that the preferred assumption is that language productivity should be viewed in terms of a vast set of sentences that are each finite, but allow all the necessary creativity and effability. We concluded that promiscuous over productivity could be curbed by replacing recursion with an iterative looping process that is limited to the depths dictated by actual language use. Finally, given that depth appears to be severely limited to two or three levels we showed that a limited process of iteration could be efficiently replaced with NEARs.

Consequences

To the extent that the argument summarized above is sound and compelling there are consequences for the controversial issues raised by Hauser et al. and listed in our introductory paragraph. Concluding that recursion is neither necessary nor preferred in models of human language production and understanding does not settle these issues because they are multi-faceted. However, it does make several conjectures less surprising and, perhaps, more compelling. First consider Everett's (2005) conclusion, endorsed by Sakel and Stapert (2010) that there are languages (viz., Pirahã) that lack "recursive structures". If recursive processing is not underlying the occurrence of "recursive structures" then it is quite reasonable to expect that some languages avoid "recursive structures". Second consider Evans and Levinson's (2009) conclusion that there are no non-trivial language universals. This claim should gain great currency for those who (up until now) believed that recursion was the only member of the FLN set. If recursion is jettisoned and replaced with standard CFG rules then there is no universal process common to all languages. This leads to Christiansen and MacDonald's (2009) related claim that there are no innate specialized mechanisms underlying language acquisition. They show that a Simple Recurrent Network (Elman 1990) can learn to recognize center-embedded sentences. The recurrent network model is incapable of true recursive processing by definition because the model is not symbolic.[8] But if human language also eschews recursive processing this strengthens, not weakens, the empirical force of their simulations. Finally, if recursion is not the core property of language, but language creativity is the bellwether for human intelligence, then we need to recast the explanation for humanity's great leap forward. It may be nothing more dramatic than memory size increase (to store the words, phrase, and grammar patterns) coupled with memory-management enhancements (to accurately generate countless combinations of the various memory components).

Conclusions

A generative aspect, realized as recursive (or looping) components within grammar rules, is not necessary to account for an extremely large and potentially unbounded

[8] We touched on this symbolic requirement earlier when, for example, we noted that a recursive grammar must have a non-terminal to recurse upon whereas an equivalent iterative grammar need not.

performance set. Inspection of the performance set reveals very little actual contribution from recursive components within rules, e.g. no deeply nested syntax nor very long lists of syntactic elements. As well as being unnecessary, recursive (or iterative) rule components are detrimental to psycholinguistic research because such a grammar necessitates a 'competence' hypothesis within which 'performance' (the real phenomenon of interest because it is the one amenable to empirical study) is swamped. A grammar based on explicit alternative syntactic structures (all the ones that commonly occur in performance sets) can define a competence set that coincides with the largest performance sets (of some language) and only slightly overproduces the performance sets of less 'competent' users of that language. Such grammars, which focus on performance, should be much better guides for research into the mind/brain mechanisms that give rise to the observed performance sets.

Appendix: Building a Parse-Tree Using the Recursive Grammar Parsing Algorithm

The following narrative makes it easier to understand how the algorithm builds the tree structure. Start with the main procedure called Streebuild and Sentence 1. The first line establishes node 1 as the root of the tree and labels it "S". Next, the procedure Npbuild(prefix, np) is called with no prefix and the np parameter taking the value of "The cat". This leads to the branch left and establishes node 2 as "The cat". Processing returns to Streebuild and the recursive Thatclbuild procedure is called. The "then" branch establishes trunk node 3 and labels it "thatcl", calls Npbuild with "that" and "the dog" which branches left, and establishes "that the dog" as node 4. Thatclbuild now recursively recalls itself, because "that" heads the remaining word-string fragment, the "then" branch establishes the trunk node, node 5 and labels it "thatcl." It now calls Npbuild with "that" and "John" which branches left establishing "that John" as node 6. Thatclbuild recursively calls itself once more. But now the number of "that"s in Sentence 1 has been exhausted and processing immediately returns by means of the "else" branch to complete the previous incomplete recursive call. This is a call to Vpbuild with "saw" which branches right to establish "saw" as node 7. Now this recursive call is complete and so control returns to previous incomplete call, i.e., it up to node 3, and calls Vpbuild with "chased" which branches right and establishes "chased" as node 8. The last call to Thatclbuild is complete so finally control returns to Streebuild where the final call to Vpbuild with the vp "bit the mouse" branches right and establishes "bit the mouse" as node 9.

References

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 3*, 113–124.

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

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

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science, 23*, 157–205.

Christiansen, M. H., & MacDonald, M. C. (2009). A usage-based approach to recursion in sentence processing. *Language Learning, 59*, 126–161.

Corballis, M. C. (2007). Recursion, language, and starlings. *Cognitive Science, 31*, 697–704.

Corballis, M. C. (2011). *The recursive mind: The origins of language, thought, and civilization*. Princeton, NJ: Princeton University Press.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*, 179–211.

Epstein, S., & Hornstein, N. (2004). Letter on 'the future of language'. *Language, 81*, 3–6.

Evans, N., & Levinson, S. (2009). The myth of language universals: Language diversity and its importance for cognitive science. *Behavioral and Brain Sciences, 32*(5), 429–448.

Everett, D. (2005). Cultural constrains on grammar and cognition in Pirahã: Another look at the design features of human language. *Current Anthropology, 76*(4), 621–646.

Gibson, E. (1998). Linguistic complexity: Locality of syntactic dependencies. *Cognition, 68*, 1–76.

Gibson, E., & Thomas, J. (1996). The processing complexity of English center-embedded and self-embedded structures. In C. Schutze (Ed.), *Proceedings of the NELS 26 sentence processing workshop* (pp. 45–71). Cambridge, MA: MIT Press.

Gibson, E., & Thomas, J. (1999). Memory limitations and structural forgetting: The perception of complex ungrammatical sentences as grammatical. *Language and Cognitive Processes, 14*, 225–248.

Harder, P. (2010). Over the top—recursion as a functional option. In H. Hulst (Ed.), *Recursion and human language* (pp. 233–244). New York, NY: De Gruyter Mouton.

Harel, D. (2000). *Computers Ltd, what they really can't do*. Oxford: Oxford University Press.

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

Huddleston, R. (1976). *An introduction to English transformational syntax*. London: Longman.

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*, 211–225.

Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension: Individual differences in working memory. *Psychological Review, 99*, 122–149.

Karlsson, F. (2010). Recursion and iteration. In H. Hulst (Ed.), *Recursion and human language* (pp. 43–67). New York, NY: De Gruyter Mouton.

Katz, J. J. (1978). Effability and translation. In F. Guenthner & M. Guenthner-Reutter (Eds.), *Meaning and translation: Philosophical and linguistic approaches* (pp. 191–234). London: Duckworth.

Kimball, J. (1973). Seven principles of surface structure parsing in natural language. *Cognition, 2*, 15–47.

Kinsella, A. (2010). Was recursion the key step in the evolution of the human language faculty? In H. Hulst (Ed.), *Recursion and human language* (pp. 179–191). New York, NY: De Gruyter Mouton.

Langacker, R. W. (1973). *Language and its structure* (2nd ed.). New York, NY: Harcourt Brace Jovanovich.

Lasnik, H. (2000). *Syntactic structures revisited: Contemporary lectures on classic transformational theory*. Cambridge, MA: MIT Press.

Laury, R., & Ono, T. (2010). Recursion in conversation: What speakers of Finnish and Japanese know how to do. In H. Hulst (Ed.), *Recursion and human language* (pp. 69–91). New York, NY: De Gruyter Mouton.

Marcus, M. (1980). *A theory of syntactic recognition for natural language*. Cambridge, MA: MIT Press.

Mithun, M. (2010). The fluidity of recursion and its implications. In H. Hulst (Ed.), *Recursion and human language* (pp. 18–41). New York, NY: De Gruyter Mouton.

Perfors, A., Tenenbaum, J., Gibson, E., & Regier, T. (2010). How recursive is language? A Bayesian exploration. In H. Hulst (Ed.), *Recursion and human language* (pp. 159–177). New York, NY: De Gruyter Mouton.

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

Pullum, G., & Scholz, B. C. (2010). Recursion and the infinitude claim. In H. Hulst (Ed.), *Recursion and human language* (pp. 113–137). New York, NY: De Gruyter Mouton.

Sakel, J., & Stapert, E. (2010). Piraha—In need of recursive syntax? In H. Hulst (Ed.), *Recursion and human language* (pp. 3–16). New York, NY: De Gruyter Mouton.

Schank, R., & Wilks, Y. (1974). The goals of linguistic theory revisited. *Lingua, 34*, 301–326.

Stabler, E. P. (1994). The finite connectivity of linguistic structure. In C. Clifton, L. Frazier, & K. Rayner (Eds.), *Perspectives on sentence processing* (pp. 303–336). Hillsdale, NJ: Lawrence Erlbaum.

Stabler, E. (1999). Formal grammars. In R. A. Wilson & F. C. Keil (Eds.), *The MIT encyclopedia of the cognitive sciences* (pp. 320–322). Cambridge, MA: MIT Press.

Tiede, H.-J., & Stoute, L. N. (2010). Recursion, infinity, and modeling. In H. Hulst (Ed.), *Recursion and human language* (pp. 147–158). New York, NY: De Gruyter Mouton.

van der Hulst, H. (2010). *Recursion and human language*. New York, NY: De Gruyter Mouton.

Verhagen, A. (2010). What do you think is the proper place of recursion? Conceptual and empirical issues. In H. Hulst (Ed.), *Recursion and human language* (pp. 93–110). New York, NY: De Gruyter Mouton.

Yang, C. (2006). *The infinite gift*. New York, NY: Scribner.