

# Psycholinguistic adequacy of left-corner parsing for Minimalist Grammars

*Anonymous*

## ABSTRACT

This paper shows that a left-corner parser for Minimalist Grammars (MGs) requires more memory resources to parse center-embedding structures than both left- and right-embeddings. The behavior of the parser mimics that of a human parser. To measure memory resources, we propose a complexity metric that is derived from the familiar notion of tenure and reliably captures the processing phenomena. The results indicate the viability of left-corner parsing for MGs as a model for human sentence processing.

*Keywords:*  
*parsing, sentence processing, Minimalist Grammars, relative clauses*

## INTRODUCTION

1

The difficulty of human sentence processing is influenced by several factors, including syntactic structure. For example, in English, a multi-layered center-embedding construction (1a) is notoriously difficult to process compared with left- and right-embedding constructions (1b and 1c) with the same number of embedded layers.

- (1) a. # The rat that the cat that the dog chased bit ate the cheese.
- b. John's brother's cat despises rats.

- c. This is the dog that chased the cat that bit the rat that ate the cheese.

(adapted from Resnik 1992)

Moreover, for center-embeddings, the number of embedded layers quickly reaches an upper limit, after which the sentence is no longer comprehensible. While linguists can construct center-embedding sentences with up to four embedded layers (e.g., *John whom June whom Paul whom Jean whom Dick hates adores prefers detests loves Mary*. Bar-Hillel 1966 cited in Karlsson 2007), naturally occurring center-embeddings, for instance, ones that are found in corpora have a maximum of three embedded layers (Karlsson 2007). Such a limit is not found in left- or right-embeddings. (1b) and (1c) can be extended to (2a) and (2b), respectively, without a drastic decrease in comprehensibility.

- (2)
  - a. John's brother's cat's neighbor despises rats.
  - b. This is the dog that chased the cat that bit the rat that ate the cheese that had eyes.

To account for this processing bias, Resnik (1992) proposes a linking theory between the observed phenomena and the parser. He shows that a left-corner (LC) parser for context-free grammars (CFGs) requires more memory to process center-embeddings compared with left- or right-embeddings. Specifically, to parse center-embeddings, the parser requires a memory storage that is proportional to the height of the syntactic tree, in contrast to only a constant size of memory storage needed to parse left- and right-embeddings, irrespective of the number of embedded layers.

Despite the elegance of the CFG-based modeling account, the formalism proves too restrictive for capturing the complexity of natural languages (Shieber 1985). To address this, subsequent studies moved to more sophisticated formalisms such as minimalist grammars (MGs, Stabler 1997). Koble *et al.* (2013) show that the offline processing difficulty contrast between center- and right-embeddings is derivable from the memory resource needed by a top-down parser for MGs to process the said structures: a center-embedding structure requires more memory resources to parse than a right-embedding one.

In Kobele *et al.* (2013), the memory cost is reliably measured using tenure, a complexity metric reflecting the amount of “time” a parse item is retained in memory. Graf *et al.* (2017) further show that a combination of tenure-based metrics derives processing bias found in subject and object relative clauses cross-linguistically. This enterprise of modeling work has since grown in both empirical coverage and the understanding of complexity metrics (e.g., verb-clusters Kobele *et al.* 2013; stacked relative clauses in Mandarin and English Zhang 2017; attachment ambiguity in English and Korean Lee 2018; gradient difficulty in Italian relative clauses De Santo 2019; end-weight preference in English and Mandarin Liu 2022, among others).

Recall that center-embedding structures are harder to process than both left- and right-embeddings. Complexity metrics in top-down MG parsing only capture the contrast between center- vs. right-embedding, but not center- vs. left-embedding (Kobele *et al.* 2013). Furthermore, based on the left-corner parser for CFG, Resnik (1992) argues that left-corner parsing is a more plausible model for the human sentence processing mechanism. A natural question is whether a left-corner parser for MGs can derive the three-way processing distinction of the embedded sentences and thus serve as a model for human sentence processing.

This paper aims to answer this question. We show that a left-corner parser for MGs indeed requires more memory resources to parse center-embeddings than both left- and right-embeddings. In doing so, We propose a complexity metric that still features the notion of tenure and makes desirable processing predictions. The results indicate that left-corner parsing for MGs is a viable model for human sentence processing.

The paper proceeds as follows. Section 2 introduces the grammar formalism, MG, its LC parser, and the notion of tenure in LC MG parsing. Section 3 presents modeling results for left-, center-, and right-embedding structures. Section 4 discusses implementational choices including syntactic assumptions, move-strategies, and other potential complexity metrics. Section 5 concludes the paper.

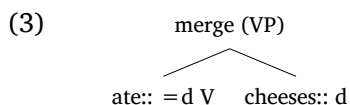
## 2 LEFT-CORNER PARSING FOR MINIMALIST GRAMMARS

The left-corner parser for MGs used in this study is based on Stanojević and Stabler (2018) and Hunter *et al.* (2019). In this section, We first introduce the grammar formalism, its parser, and a tree annotation scheme to represent the parser’s behavior using derivation trees. Based on tree annotations, We then discuss complexity metrics focusing on tenure – the amount of time a parse item is stored in memory.

### 2.1 *Minimalist Grammar and its left-corner parser*

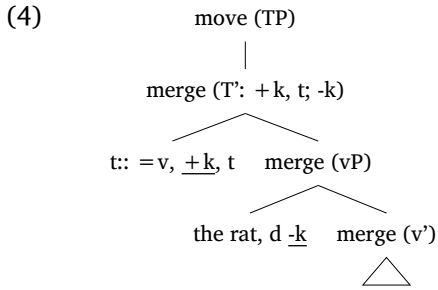
Minimalist Grammar (MG, Stabler 1997, 2011) is a lexicalized, context-sensitive grammar formalism based on the Minimalist Program (Chomsky 2014). A lexical item in MGs is a feature bundle that expresses information including its sound, category, and dependency relations. Category and dependency features enter feature-checking relations via operations *merge* and *move*, according to which lexical items combine into sentences (trees).

*Merge* combines lexical items and/or derived categories. For example, lexical items *ate* and *cheeses* can be merged to build a VP such as (3).



(3) is a derivation tree that records the process of structure-building. In (3) *ate* has a selection feature = d and a category feature V. It merges with *cheeses* that has a matching category feature d. This merge checks the d features and builds a VP (which carries the category feature V). It is then ready to merge with a lexical item that selects a VP (i.e., has a selection feature =V).

*Move* extends the existing tree to include the landing site of the movement. For example, in a sentence such as *The rat ate cheeses* the subject moves due to the extended projection principle (EPP). This movement is licensed by matching case features k on the mover *the rat* and the licensor *t*. The derivation is shown in (4).



In (4) the T' node is a derived category containing features +k, t from the T head and -k from the mover which percolates up. The *move* operation checks the matching k features, extends the tree with a node carrying the remaining feature t. This is the TP node, the landing site of the EPP movement.

Derivation trees like (4) differ from phrase structure trees generated by CFGs in that the order of leaves (terminal nodes) may not correspond to the linear order of words in the sentence. This distinction is important when interpreting modeling results, especially when derivation trees and phrase structure trees happen to have the same leaf order. We will discuss this in more detail in Section 4.1.

The left-corner parser for MGs defined in Stanojević and Stabler (2018); Hunter *et al.* (2019) allows for sound and complete parsing for MGs.

The parser keeps track of four pieces of information:

- the current position in the string
- a distinguished “top” item in the parser’s storage
- “slotted” items which are phrases conjectured based on the left-corner prediction but yet to be fully built (phrases with an open slot in them)
- built items which are fully built phrases that become the current top item for further operations.

Following the notation in Hunter *et al.* (2019), a parse item takes the form of a single-node item or an implication item, examples of which are found in (5) and (6), respectively.

(5)  $((i, j) \cdot C), M$

(6)  $((i, j) \cdot C), M \Rightarrow ((k, l) \cdot K), N$

A single-node item such as (5) states that the string span between position  $i$  to  $j$  corresponds to a category  $C$ . The indices mark the position between each input word. The dot (“.”) is a placeholder for “:.” or “:.” which denotes whether the category following the symbol is lexical or derived.  $M$  and  $N$  are variables for mover queues. Individual items in a mover queue take the form of a single-node item.

An implication item is an implication from one single-node item to another, hence the use of the implication arrow “ $\Rightarrow$ ”. An implication item such as (6) means that if in the string span  $i$ - $j$  the parser finds a category  $C$ , it can confirm that the span  $k$ - $l$  corresponds to a category  $K$ . Implication items are slotted items stored after left-corner predictions.

For our purpose, we omit the string span and mover queues in our parse item expression and use tree nodes and lexical items rather than categories. This abstraction increases readability and obtains a straightforward correspondence between parsing traces and derivation trees – at the cost of being unable to distinguish potential choice points during a parse. Since our purpose is to model offline processing difficulties and highlight the role of syntactic structures, we can assume that the parser always constructs the correct structure. Therefore, this simplified notation will suffice. We include both the full notation and the simplified notation of parse items in the following paragraphs discussing parser operations and stick to the simplified one thereafter.

At each parsing step, the operations available to the parser are *shift*, *connect*, *complete*, *LC predict*, and *unmove*. *Shift* reads in the next word  $W$  from the input and stores as the top item a single-node item of the form  $((i, i+1) : : W)$  (or  $((i, i) : : W)$  in case  $W$  is an empty string, which the simplified notation cannot distinguish).

The *connect* operation is unique to an arc-eager variant of the parser, according to which the parser is able to connect a newly created parse item to existing ones. Stanojević and Stabler (2018); Hunter *et al.* (2019) list the following variants of connections:

- (7) Connecting operations (notation adapted from Hunter *et al.* 2019)
- a. If  $B$  is the newly created item and  $B \Rightarrow A$  an existing one, then connection produces  $A$ .

- b. If  $B \Rightarrow A$  is the newly created item and  $C \Rightarrow B$  an existing one, then connection produces  $C \Rightarrow A$ .
- c. If  $C \Rightarrow B$  is the newly created item and  $B \Rightarrow A$  an existing one, then connection produces  $C \Rightarrow A$ .
- d. If  $C \Rightarrow B$  is the newly created item and  $B \Rightarrow A$  and  $D \Rightarrow C$  existing one, then connection produces  $D \Rightarrow A$ .

The operations in (7b-7d) correspond to connections where the newly created item is the bottom or top of an existing item or where it connects two items on both ends. All connection operations remove from the storage items that are used for connection and stores as the top item the new item produced by connection.

In departure from Stanojević and Stabler (2018) and Hunter *et al.* (2019), We highlight (7a) as a separate operation, *Complete*, which applies to both arc-eager and arc-standard strategies. *Complete* removes an implication item from memory when the single-node item on the left of the implication arrow is found. The item on the right of the arrow now becomes the top item, ready for further operations.

*LC predict* and *unmove*, which are crucial operations for the LC parser, are discussed now with examples. *LC predict* creates an implication item based on the left-corner of a subtree. Consider the previous merge example in (3) which creates a VP *ate cheeses*. Given a grammar that allows this merge, *ate* is the left-corner of the VP. If *ate* is the current top item, the parser produces an implication item like (8).

- (8) DP  $\Rightarrow$  VP  
 C.f.,  $((k, j) \cdot d) \Rightarrow ((i, j) :V)$  (assuming that *ate* is between position *i* and *k*)

(8) is an implication from the verb's complement, DP, to its parent VP. It states that if the parser can find a DP, it can confirm the existence of its parent VP. The full notation contains the same core information with the categories. The string spans show that based on the input *ate* between *i* and *k*, the parser makes predictions on the position immediately after it, namely, *k* to *j*, and on the category of a larger span, *i* to *j*, that corresponds to the entire VP structure.

Finally, the *unmove* operation constructs the landing site of a movement. We follow the move-eager strategy described in Hunter *et al.* (2019), where the parser can immediately apply *unmove* upon

encountering the licensing head of the movement. In the example provided in (4),  $t$  licenses EPP movement. When  $t$  is processed, the parser produces the following:

- (9) a.  $vP \Rightarrow T'$   
 C.f.,  $((i, j):v), ((k, l):-k) \Rightarrow ((i, j):+k t), ((k, l):-k)$   
 (assuming that  $t$  is at position  $i$  and the move has been found and processed earlier between position  $k$  and  $l$ )
- b.  $T' \Rightarrow TP$   
 C.f.,  $((i, j):+k t), ((k, l):-k) \Rightarrow ((k, j):t)$
- c. (combining 9a and 9a)  $vP \Rightarrow TP$

Let us first unpack the full notation in (9a). This parse item indicates a left-corner prediction based on a  $t$  node at position  $i$ . It means that if from the string span immediately after  $i$  the parser finds a  $vP$  with a mover inside it moving for  $k$ , the parser can confirm a node that would become a  $TP$  after the movement for  $k$  is licensed. This is a  $T'$ . The mover inside  $vP$ , which is assumed to be found earlier at position  $k-l$ , carries over to the  $T'$  node.

The full notation in (9b) indicates the application of a *move* operation discussed earlier in (4). As allowed by the move-eager strategy, the parser can take the mover,  $((k, l):-k)$ , to be the one that satisfies the  $k$  feature left in the  $T'$  node. By applying *move*, the landing site,  $TP$ , is created right away. The beginning of the string span is updated to  $k$  to cover the early encountered mover.

The simplified notations indicate the above operations using tree nodes. The parse items are further simplified as (9c) since the internal structure,  $T'$ , is considered found by the parser.

Hunter *et al.* (2019) posit the need for the move-eager strategy to model active gap-finding but nothing technical seems to hinge on this. We address this in Section 4.2 and show that incorporating immediate extensions into our model does yield desirable processing predictions.

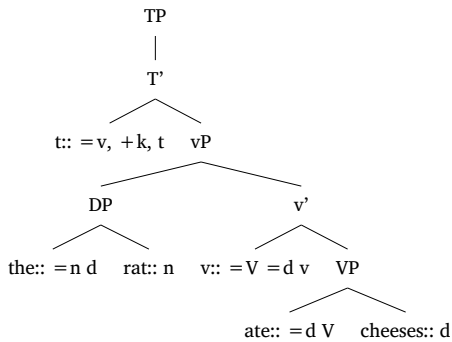
Putting together, (10) shows the steps of an arc-eager LC MG parser building the derivation tree in (11) for the sentence *The rat ate cheeses*.

(10)	Step	parse item
	1. shift the::	the::



2. LC the:: <hr/>	NP ⇒ DP
3. shift rat:: <hr/>	DP:
4. LC the rat: <hr/>	v' ⇒ vP
5. shift t:: <hr/>	t:: v' ⇒ vP
6. LC t:: <hr/>	v' ⇒ TP
7. shift v:: <hr/>	v:: v' ⇒ TP
8. LC v:: <hr/>	VP ⇒ TP
9. shift ate: <hr/>	ate:: VP ⇒ TP
10. LC ate:: <hr/>	DP ⇒ TP
11. shift cheeses:: <hr/>	TP

(11)



The first five steps in (10) should look unsurprising. We examine step 6 in more detail. A *LC predict* followed by *unmoved* based on *t* first produces  $vP \Rightarrow TP$ , as discussed earlier. Given the arc-eager specification, the parser is able to immediately *connect* newly created parse items with structures already built. Therefore,  $vP \Rightarrow TP$  *connects* with  $v' \Rightarrow vP$  created at step 4, resulting in  $v' \Rightarrow TP$ .

This ability to *connect* items immediately is unique to arc-eager parsers. In contrast, an arc-standard parser would instead store  $vP \Rightarrow$

TP as a separate parse item at step 6. As shown by Resnik (1992), the choice of arc-strategy impacts the processing predictions of a left-corner parser for CFGs. We will see in Section 3.4 that this holds true for a LC MG parser.

Also, in (10) the parse item  $v' \Rightarrow vP$  is stored in memory at steps 4 and 5, for a total of two steps. This is the tenure of the parse item. Next, we explore the role of tenure as the basis for complexity metrics in LC MG parsing.

## 2.2

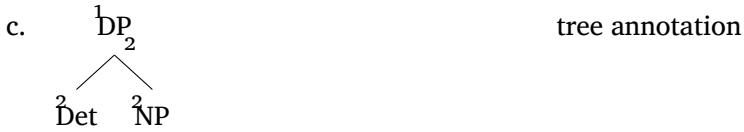
### *Tenure-based complexity metric*

Tenure, one of the key measurements of memory resources in top-down MG modeling work, applies straightforwardly to left-corner MG parsing. For left-corner parsing, the tenure of a parse item is still the number of steps it is retained in memory. From a trace of the parser's progression such as (10), one can already calculate the tenure for each parse item and make processing predictions. For example, the parse item  $v' \Rightarrow vP$  has a tenure of 2: it is stored in memory at steps 4 and 5.

While it is straightforward to calculate tenure from a parsing trace, it can be humanly challenging to reconstruct from the parse items the syntactic tree built at a given step. Representing parse items and parser behaviors in trees helps visualize how tree geometry affects parsing. This is when annotated derivation trees can be helpful.

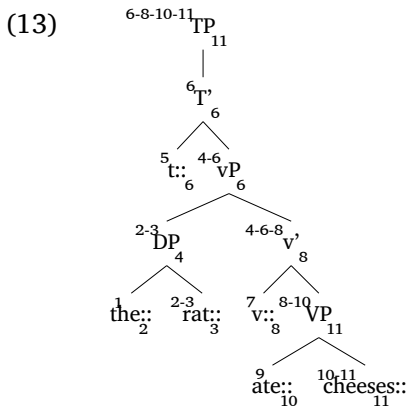
Tree annotations are central to modeling work using top-down MG parsers. Annotated derivation trees are condensed yet complete representations of a parse. They allow us to track the parser's traversal through the tree structure and calculate the associated memory usage. In top-down MG parsing, representing parse items using tree annotations is uncomplicated because the tree nodes have one-to-one correspondences to the parse items. For example, consider a top-down parse that conjectures a DP node and from it derives determiner (Det) and NP.

- (12) a.  $\frac{DP}{Det \quad NP}$  parser rule  
 b. Step 1. DP memory stack  
 Step 2. Det | NP



(12a) is the parser rule for top-down prediction<sup>1</sup>, indicating that Det and NP are derivable from DP. In (12b), the parser predicts and stores DP at step 1. At step 2, by applying (12a), the parser removes DP, predicts and stores Det and NP. In (12c), the superscript (index) indicates the step the item/node enters the memory storage, the subscript (outdex) the step the item/node exits the storage.

LC MG parsing differs from top-down MG parsing in that a parse item does not always correspond to a single node in the derivation tree. Furthermore, a node in the tree can be predicted multiple times throughout LC MG parsing. To faithfully represent the behavior of the LC parser, we need to adjust the tree annotation scheme. (13) illustrates how annotation works for a LC MG parse of the familiar sentence *The rat ate cheese*.



First, in LC MG parsing a parse item corresponds to a single node if it is a leaf node or a newly completed node. In the case of a leaf node, the index records the step when the parser *shifts* it into memory. If it is a newly completed node, the index records the *complete* step. The

<sup>1</sup> The rule notation is adapted from Kobele *et al.* (2013). The current adaptation makes predictions based on binary merges while the original rule is more general.

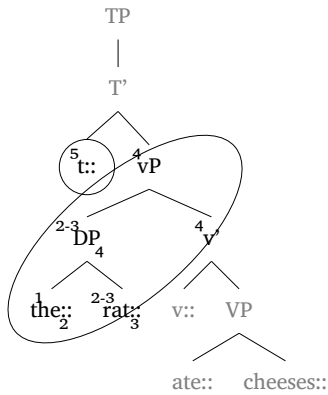
outdex for a single node, on the other hand, indicates the step when the node/item exits memory as it is used in any parser operation. For example, the leaf node *the* is introduced at step 1 as the parser *shifts* and exits memory at step 2 when used for a *LC prediction*. It has an index of 1 and an outdex of 2.

Next, when a parse item represents an implication, it corresponds to two nodes in the derivation tree. The indices on the nodes record the steps at which the nodes are part of a parse item. If a node is part of a strictly different parse item at a different step in the parsing process, we record the step in the index and connect the recorded steps with a dash (“-”). The outdex records the step when a node is no longer part of any implication.

For example, in (13), the vP node is first predicted by a *LC prediction* based on its daughter DP at step 4. The same index is found on the *v'* node, which belongs to the same parse item  $v' \Rightarrow vP$  that can be found in (10) at step 4. This vP node is predicted a second time based on its sister *t* at step 6. It is now part of a different parse item  $v' \Rightarrow vP$ . Thus, the index of the vP node has 4-6. Also at step 6, due to the *connection* mentioned earlier, the vP node is no longer part of any parse item. The outdex of vP is therefore 6.

This tree annotation scheme allows for exact reconstructions of the parse items at each step. At any given step, the parse items are subtrees that are annotated up to that step. For example, at step 5, two subtrees are built, as circled in (14). The first subtree is the vP with un-outdexed vP and *v'* nodes. The other subtree is the un-outdexed *t*. These two subtrees correspond to the two parse items  $t$  and  $v' \Rightarrow vP$  found in (10) at step 5.

(14)



Moreover, by looking for the matching dash-connected index pair on two nodes, the parse item stored between these two steps can be unambiguously recovered. And by then taking the difference of the two steps, we have *item tenure*, the duration that the parse item is stored in memory<sup>2</sup>. For example, the vP and v' nodes in (13) have matching dash-connected indices of 4-6. The parse item these two nodes consist of is  $v' \Rightarrow vP$ , carrying an *item tenure* of 2 ( $=6-4$ ).

Just as in top-down MG parsing, one can explore a variety of complexity metrics based on *item tenure*. Here we identify just one such possibility:  $\text{MaxT}_{item}$ , which is the maximal duration that any item remains in memory. The  $\text{MaxT}_{item}$  of a parse can be calculated solely from the annotated derivation tree: it is the maximum difference among all dash-connected indices on tree nodes. For example, the  $\text{MaxT}_{item}$  of (13) is 2 found on multiple items.

$\text{MaxT}_{item}$  is chosen as the candidate for the complexity metric for two reasons. First, its top-down MG parsing counterpart,  $\text{MaxT}$ , has been shown to reliably capture processing difficulties (Kobele *et al.* 2013; Graf *et al.* 2017, a.o.).  $\text{MaxT}_{item}$  is a natural first candidate to consider in LC MG parsing. Second, given the current annotation

<sup>2</sup>This *item tenure* calculation only applies to parse items that correspond to two nodes. Given the current parser setup, items that correspond to one node remain in memory for one step. Then they are either removed from memory at the same step for *complete*, or at the next step for *LC prediction*. We thus ignore these parse items in the memory cost calculation.

scheme, calculating the maximum of *item tenure* requires only a visual check on the nodes.

The results of our model, to be discussed next, suggest that  $\text{MaxT}_{item}$  as the complexity metric correctly predicts the processing difficulty difference between left-, center-, and right-embeddings. It is also sensitive to arc-strategies and makes desirable predictions.

## 3

## RESULTS

The processing phenomena to be modeled are the processing difficulties of left-, center-, and right-embedding structures. A total of six target sentences, found in (15) - (17), are included in the analysis.

- (15) Left-embedding
- a. The rat’s cheese is here. 1-layer
  - b. The rat’s cheese’s eyes are missing. 2-layer
- (16) Center-embedding
- a. The rat that the cat bit is here. 1-layer
  - b. The cheese that the rat that the cat bit ate is here. 2-layer
- (17) Right embedding
- a. The rat that ate cheeses is here. 1-layer
  - b. The rat that ate the cheese that had eyes is here. 2-layer

For each target sentence, two arc-strategies are included. The total number of LC MG parses is 12 (3 directions  $\times$  2 layer conditions  $\times$  2 arc-strategies). For each embedding direction, we compare the  $\text{MaxT}_{item}$  of 1-layer and 2-layer sentences for both arc-eager and arc-standard parses. The overall results are in Table 1.

Table 1:  
Modeling results  
based on  
 $\text{MaxT}_{item}$

	$\text{MaxT}_{item}$		
	Left	center	right
1-layer <sub>arc-eager</sub>	2	10	6
2-layer <sub>arc-eager</sub>	2	24	6
1-layer <sub>arc-standard</sub>	4	15	13
2-layer <sub>arc-standard</sub>	4	29	27

Parser	Left	center	right
LC <sub>MG</sub> (arc-eager)	$O(1)$	$O(n)$	$O(1)$
LC <sub>MG</sub> (arc-standard)	$O(1)$	$O(n)$	$O(n)$
LC <sub>CFG</sub> (arc-eager)	$O(1)$	$O(n)$	$O(1)$
LC <sub>CFG</sub> (arc-standard)	$O(1)$	$O(n)$	$O(n)$
Human parser	$O(1)$	$O(n)$	$O(1)$

Table 2:  
Modeling results  
in big-O  
notation. Table  
format, CFG and  
Human parser  
results (marked  
in gray) from  
Resnik 1992.

The parsing complexities under different conditions can be summarized using big-O notations (Resnik 1992), which is found in Table 2.

Overall, the current model successfully captures different human parsing difficulties between the three embeddings. Using  $\text{MaxT}_{item}$  as the complexity metric, left- and right-embeddings exhibit constant memory costs as the number of layers increases. In contrast, center-embeddings show a memory load increase that is proportional to the number of layers. Furthermore,  $\text{MaxT}_{item}$  exhibits sensitivity to arc-strategies. We next look at the results from each embedding direction, followed by a discussion on arc-strategies.

### Left-embedding

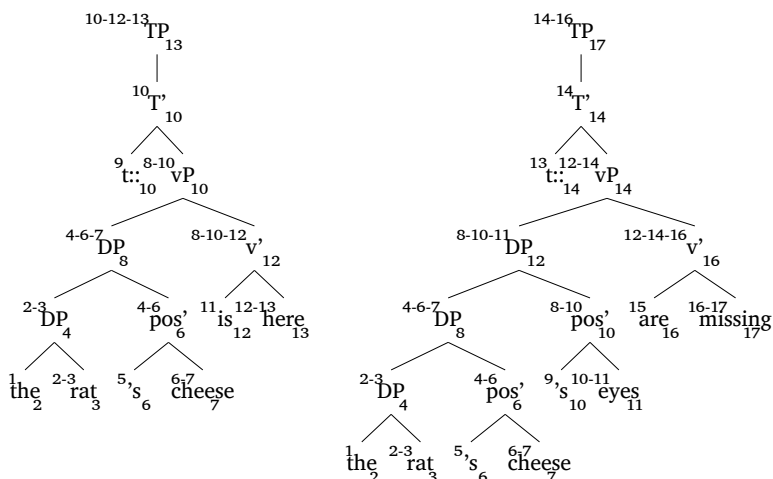
3.1

The target sentences for left-embedding are (18) and (19). Following a functional category analysis of the possessive head (Adger 2003), the “b” sentences are the leaf nodes of the target sentences, which the parser takes as input strings.

- (18) a. The rat’s cheese is here.  
b. the rat ’s cheese t is here
- (19) a. The rat’s cheese’s eyes are missing  
b. the rat ’s cheese ’s eyes t are missing

Under the arc-eager strategy, tree annotations for (18) and (19) are found in Figure 1.  $\text{MaxT}_{item}$  remains constant at 2 for both layer conditions (and at 4 for arc-standard parses). This suggests that, as the number of layers increases, the parser only requires a constant memory space to process left-embeddings.

Figure 1:  
(Arc-eager) tree  
annotations for  
left-embeddings



This constant memory load does not follow from the behavior of a top-down parser. We briefly pause here to see why. Assuming a correct parse, a top-down parser would prioritize the branch containing the first word at each top-down prediction. In doing so, it needs to keep track of the sister nodes on the other branch at each prediction point. This is memory costly, and the memory cost grows with the depth of the first word (i.e., the number of layers). In contrast, a LC parser avoids this memory strain by using bottom-up information directly from the input. As shown in Figure 1, regardless of the first word's depth, the parser can use it early at step 2 as the left-corner to efficiently build structures.

It is worth noting that the current syntactic assumption does not involve movement within the DP, making the structure indistinguishable from one derived by CFGs. It is thus unsurprising to see similar processing predictions based on a LC parser. As hinted earlier, We discuss an alternative structure in Section 4.1.

### 3.2

#### *Center-embedding*

The target sentences for center-embedding are the following.

- (20) a. The rat that the cat bit is here.  
b. the d-rel rat that the cat t v bit t is here



- (21) a. The cheese that the rat that the cat bit ate is here.  
b. the d-rel cheese that the d-rel rat that the cat  
t v bit t v ate t is here

The “b” sentences are the leaf nodes (input string) following a promotion analysis for relative clauses (Kayne 1994).

Assuming an arc-eager LC MG parse, tree annotation excerpts for the target sentences are in Figure 2. For the 1-layer center-embedding,  $\text{MaxT}_{item}$  is 10, found on *bit* and the VP node (shaded), which corresponds to the parse item  $V \Rightarrow VP$ . We can verify the result against the input string (20b). After reading the third word, *rat*, the parser builds the relativized object DP, makes a *LC prediction* from it, and stores  $V \Rightarrow VP$ . This item is held in memory until the VP node is predicted again as the sister of *v* which appears near the end of the input string.

For the 2-layer center-embedding,  $\text{MaxT}_{item}$  is 24, found on *ate* and the VP node (shaded), corresponding to the parse item  $V \Rightarrow VP$ . With a higher  $\text{MaxT}_{item}$  of 24 compared to the 1-layer case, the model predicts that the 2-layer center-embedding is more difficult to parse for the LC parser.

By comparing the two layer conditions, we can see that the  $\text{MaxT}_{item}$  grows with the number of embedding layers for center-embeddings.  $\text{MaxT}_{item}$  is associated with the VP node which is part of two different parse items. First, at step 6, it is in  $V \Rightarrow VP$  created by a *LC prediction* based on the relativized object. Then, at step 30, it is in  $VP \Rightarrow v'$  created by a *LC prediction* based on *v* and exits memory at the same step due to connection. Between the two steps, the parser needs to hold the item  $V \Rightarrow VP$  in memory while it first builds the subject DP. Since the embedding layers are within the subject DP, the more layers there are, the longer the hold. This is how  $\text{MaxT}_{item}$  grows with the number of embedding layers.

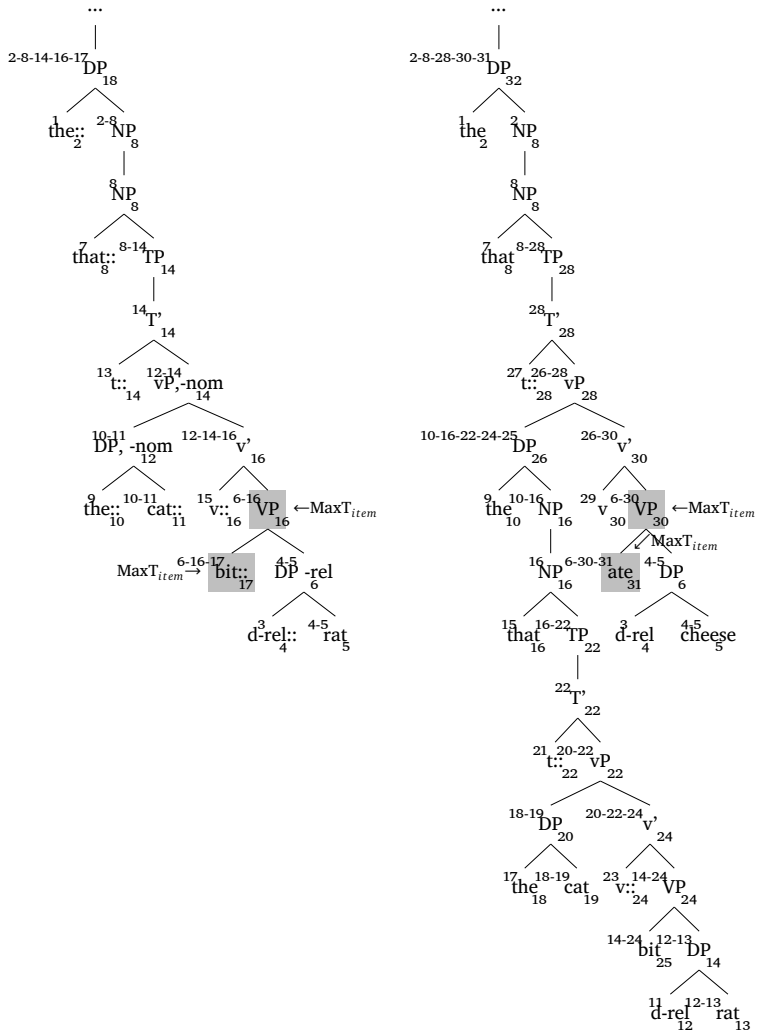
### Right-embedding

3.3

Finally, the two target sentences for right-embeddings are in (22) - (23). Same as before, the “b” sentences are the leaf nodes assuming a promotion analysis for relative clauses.

- (22) a. The rat that ate cheeses is here.

Figure 2:  
(Arc-eager) tree  
annotations for  
center-  
embedding



- b. the d-rel rat that t v ate cheeses t is here
- (23) a. The rat that ate the cheese that had eyes is here.
- b. the d-rel rat that t v ate the cheese that t v had eyes t is here

$\text{MaxT}_{item}$  for both layer conditions is 6, which predicts that the parser only needs a constant amount of memory resources to process right-embeddings. We see why in the tree annotation excerpts in Figure 3. For both layer conditions, the longest-held nodes are the NPs (shaded). The NP nodes are part of the parse item  $\text{NP} \Rightarrow \text{DP}$  created by a *LC prediction* based on *the*. The parser holds the item in memory until it can *connect* to the lower structure. Meanwhile, the parser builds the subject (and the relative pronoun). In right-embeddings, the subject does not increase in complexity as the embedding layers increase. This results in a constant hold time for the parser. Therefore,  $\text{MaxT}_{item}$  predicts that only a constant amount of memory resources is needed to process right-embeddings.

Note that there is another node bearing  $\text{MaxT}_{item}$  in the 2-layer tree in Figure 3, namely, the more deeply embedded NP node. This node is held in memory for the same reason as the topmost NP: the node is predicted and held in memory while the parser operates on its sister and is removed from memory after the parser finishes building the subject and the relative pronoun. While the maximum of *item tenures* does not increase with the number of layers, a highly tenured node found in the embedded layer reflects a general increase in memory cost associated with additional layers. We will see the effect of this in other complexity metric candidates in Section 4.4.

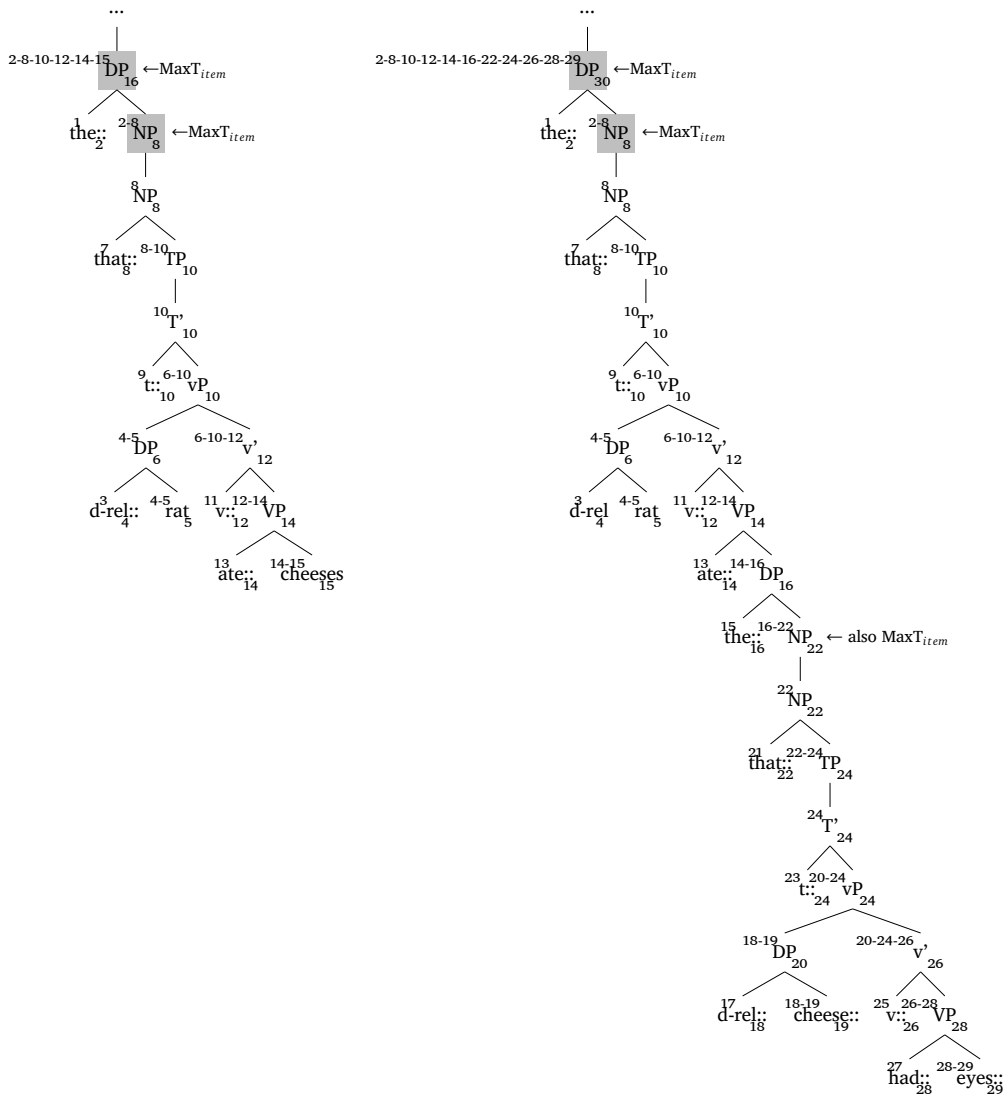


Figure 3: (Arc-eager) tree annotations for right-embedding

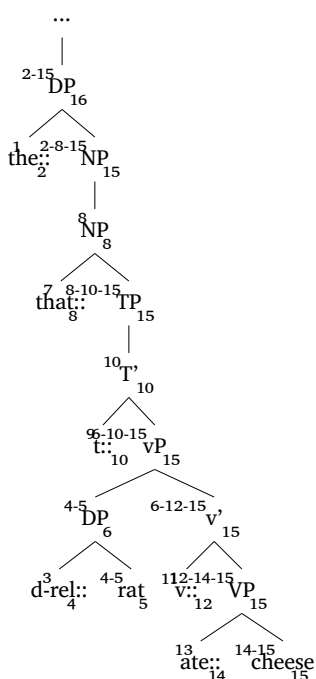
The effect of arc-strategy is captured by  $\text{MaxT}_{item}$ . This is most straightforwardly seen in the right-embeddings. In the 1-layer right-embedding tree built by an arc-eager parser, as shown on the left in Figure 3, the topmost DP node has 2-8 in its index. It is first predicted at step 2 as part of the item  $\text{NP} \Rightarrow \text{DP}$ . At step 8, the parser operates as follows:

(24) Step	parse item (w/ index)
...	... ${}^2\text{NP} \Rightarrow {}^2\text{DP}$
8.1 LC that::	... $\text{TP} \Rightarrow {}^8\text{NP}$ ${}^2\text{NP} \Rightarrow {}^2\text{DP}$
8.2 unmove	... $\text{TP} \Rightarrow {}^{2-8}\text{NP}$ ${}^2\text{NP} \Rightarrow {}^2\text{DP}$
8.3 connect at NP	... $\text{TP} \Rightarrow {}^{2-8}\text{DP}$

Step 8.1 is a *LC prediction* based on the promotion movement licenser *that*, followed by *unmove* at step 8.2, creating the parse item  $\text{TP} \Rightarrow \text{NP}$ . Crucially at step 8.3, the parser is able to *connect* this newly built item to the existing  $\text{NP} \Rightarrow \text{DP}$ , creating  $\text{TP} \Rightarrow \text{DP}$ . The DP node now becomes part of a different parse item. This change is reflected by 2-8 in its index, from which we find  $\text{MaxT}_{item} = 6$ .

In contrast, an arc-standard parser cannot connect a newly built parse item to existing ones. This affects processing predictions as can be seen from the tree annotation excerpt from an arc-standard parser (25).

(25)



In (25), the topmost DP is predicted similarly at step 2 as part of the parse item  $\text{NP} \Rightarrow \text{DP}$ . According to the arc-standard specification, the predicted NP node is considered found only when the entire subtree it dominates is built. This means the parse item  $\text{NP} \Rightarrow \text{DP}$  remains in memory until step 15 when the parser fully builds the NP and completes DP. This is where we find the  $\text{MaxT}_{item}$  of 13 (= 15-2), which is larger than that in the arc-eager parse.

Moreover, because the arc-standard parser waits to complete the entire subtree before integrating it to the topmost DP,  $\text{MaxT}_{item}$  which associates with the DP grows proportionally with the number of embedding layers in right-embeddings. This prediction aligns with that of an arc-standard LC CFG parser but is not how humans behave (Resnik 1992) (see arc-standard tree annotations for all embedding conditions in Appendix A.1).

## DISCUSSION

4

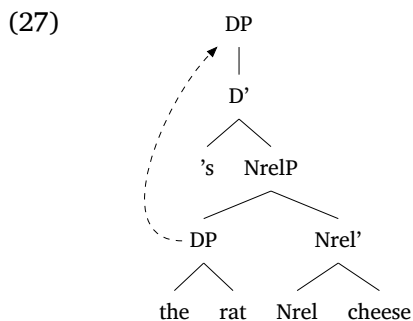
*Left-embedding: an alternative structure*

4.1

The processing predictions made by the current LC MG parser are similar to those made by a LC parser for CFG. As briefly mentioned in Section 3.1, this is to be expected for left-embeddings. In the current model, the syntactic analysis lacks any movement. As a result, the derivation tree derived by MGs is structurally identical to one that is derived by CFGs. Here we explore a movement-based analysis for left-embeddings and see if this affects the processing predictions.

The movement-based analysis to derive left-embeddings, or possessive structures, stems from the proposals treating the possessor as the internal argument of the possessee. Concretely, a 1-layer left-embedding such as (18), here repeated as (26), has a movement-based analysis schematized in (27).

(26) [The rat's cheese] is here.



This particular implementation is based on Kobele (2021). The lexical item *Nrel* turns a noun phrase into a relational noun phrase (NrelP). The DP that stands in possessive relationship to the noun phrase merges at spec NrelP and subsequently raises to the spec DP.

Under this analysis, the parser still predicts a constant memory load when the number of embedded layers increases. Both 1-layer and 2-layer left-embeddings have a  $\text{MaxT}_{item}$  of 2 assuming an arc-eager parse (tree annotations in Appendix A.2).

This eliminates the potential confound of a CFG-like analysis for left-embeddings, which might obscure the model's prediction.

The current LC MG parsing model employs a move-eager strategy as outlined by Hunter *et al.* (2019), where the parser immediately constructs the landing site upon encountering the movement licenser. Although the original proposal does not technically mandate this move-eagerness, the strategy influences processing predictions, as measured by  $\text{MaxT}_{item}$ .

Hunter *et al.* (2019) model the active-filler strategy by specifying connection preferences in a LC MG parser. The active-filler strategy refers to the human processing tendency to hypothesize gap positions actively at the first available position. Evidence for this processing tendency involves the reading time for sentences such as (28). In (28), there is a reading slowdown at the potential gap position, *books* (Stowe 1986, cited in Hunter *et al.* 2019).

(28) What John buys *books* about yesterday?

Assuming the active-filler strategy, the human parser predicts a gap for *what* at the object position of *buy*, which is the first available gap position. When the actual object *books* is later read and fills this gap, backtracking is required, causing the reading slowdown.

To model this, Hunter *et al.* (2019) contrast a LC MG parse of (28) with that of (29):

(29) What John buys?

When building structures for (28) and (29), the parser takes the same initial steps, which include building the base-merge position and landing site of *what*, namely, the  $V'$  and CP. When the parser reaches *John* and *LC predicts* based on it, a correct parse for (29) is to *connect* the existing  $V'$  as the sister of *John*, building (30). This amounts to conjecturing the gap position of *what*.

(30)

```

graph TD
  VP --- John
  VP --- V_prime[V']
  V_prime --- V
  V_prime --- what
  
```

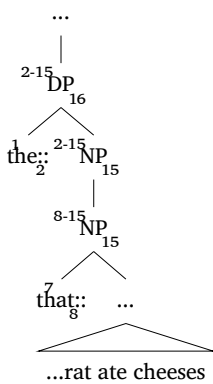


In contrast, to correctly parse (28), the V' containing *what* needs to be disconnected after the parser processes the input *John*. If a parser preferred *connection* at this point while the actual input sentence is (28), later backtracking is expected. This models the reading slow-down.

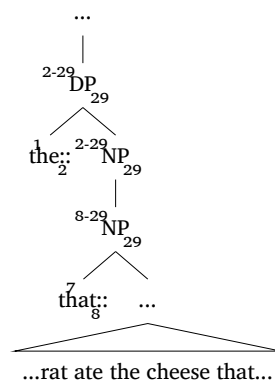
Specifying the parser's preference for *connection* is independent of whether the landing site CP is built. As Hunter *et al.* (2019, f.n. 8) point out, the “move-standard” parser of Stanojević and Stabler (2018), which builds the landing site only when everything it dominates is built, also allows choices between connection preferences. However, the timing of building the landing site (*unmove*) has consequences on processing predictions which are captured by  $\text{MaxT}_{item}$ .

Take the right-embedding parses for example. As discussed in Section 3.3, a move-eager parser predicts a constant memory cost across layer conditions. The same prediction does not hold for a move-standard parser. This can be seen in the tree annotation excerpts in (31) and (32).

(31)



(32)



(31) illustrates part of a 1-layer right-embedding parse following a move-standard strategy. When the parser processes *that* which licenses the promotion movement, it *LC predicts* based on *that*, predicting the lower NP. Due to the move-standard strategy, the parser cannot immediately *unmove* based on the lower NP. As a result, the NP cannot be correctly connected to the higher structure built earlier. Building the landing site and connecting it to the higher structure is only possible at step 15 as the entire subtree is built. This results in a

long, 13-step (= 15-2) wait for the item NP  $\Rightarrow$  DP, which is where we find  $\text{MaxT}_{item}$ .

For the 2-layer sentence in (32), the  $\text{MaxT}_{item}$  is a larger value of 27 found on the same parse item. This predicts an increased memory cost as the number of embedding layers increases. The larger  $\text{MaxT}_{item}$  is caused by a longer wait of the tenured item as the parser builds a larger subtree than in the 1-layer tree. In fact, this predicts a memory cost growth that is proportional to the number of embedding layers, which is not true for human parsers.

## 4.3

*Apparent difficulties with  $\text{MaxT}_{item}$* 

The center-embedding sentences we examined are those of object relative clauses<sup>3</sup>. In terms of processing difficulties, they contrast with stacked object relative clauses, which also involve multiple object relative clauses, but are less difficult to process as the number of relative clauses increases. Consider (33).

## (33) Stacked object relative clauses

- a. The cheese [that the rat ate] [that the cat lost] is here.
- b. The cheese [that the rat ate] [that the cat lost] [that the dog wanted] is here.

As the number of *that* clause increases from two to three, the sentence is still reasonably easy to process, unlike center-embeddings where the upper limit of embedding layers is three.

Assuming that the intuition is true that humans process stacked object relative clauses differently from center-embeddings, we can use the current processing model to evaluate syntactic proposals for stacked relative clauses, based on whether their corresponding  $\text{MaxT}_{item}$  encounters difficulties predicting the processing phenomenon. To demonstrate, We compare two such syntactic proposals, namely, the promotion analysis and an adjunction analysis. The results suggest

---

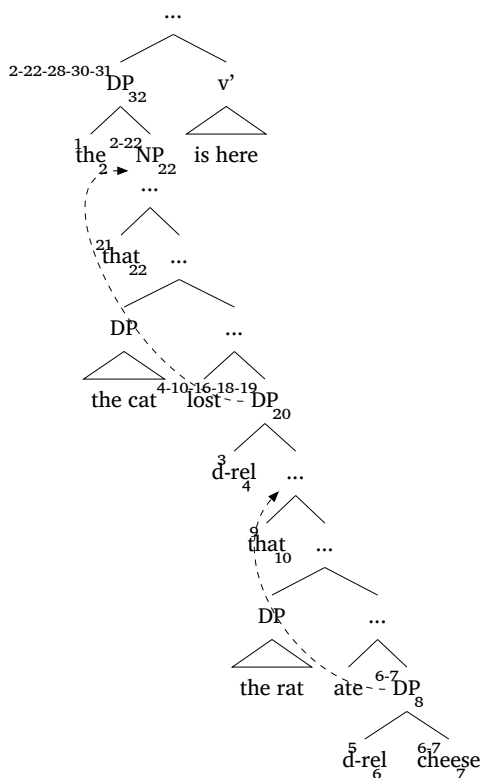
<sup>3</sup>Center-embeddings of sentential subjects, e.g., *that that the world is round is obvious is dubious*, are also “almost unintelligible” (Kuno 1974, 119). Although not discussed in this paper,  $\text{MaxT}_{item}$  in the current model predicts the processing difficulty of center-embeddings of sentential subjects, too.

that the processing difficulty follows from the syntactic structure of the adjunction analysis but not the promotion analysis.

First, based on a promotion analysis for relative clauses, as we did for embedded sentences, the processing difficulty of stacked object clauses is unexpected according to  $\text{MaxT}_{item}$ . The  $\text{MaxT}_{item}$  of a two-clause stacked object relative clause is 20, compared to 34 of a three-clause counterpart. This predicts that as the number of stacked clauses increase, the sentence becomes more difficult to process. Upon closer examination of the tree annotations, we can verify that the increase in difficulty, similar to that of center-embeddings, is proportional to the number of clauses.

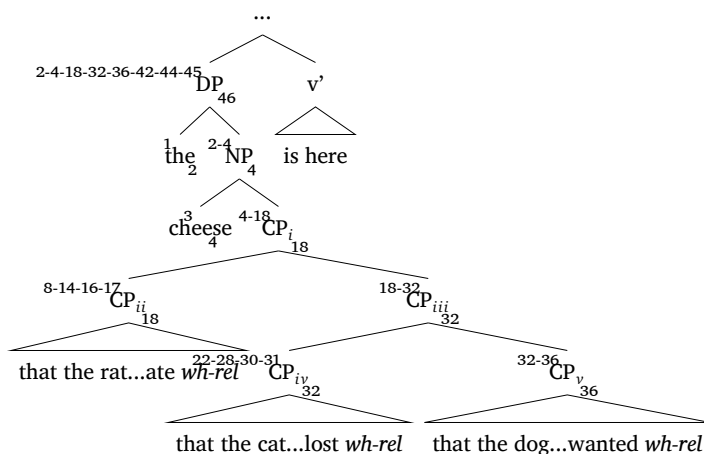
The tree annotation excerpt in (34) is from a parse for sentence (33a) assuming the promotion analysis (full tree annotations can be found in Appendix A.3 Figure 8).  $\text{MaxT}_{item}$  is found on the topmost DP and NP nodes, which correspond to the parse item  $\text{NP} \Rightarrow \text{DP}$ . This parse item is created by a LC prediction based on the first word *the*. The item remains in memory until the parser processes the second *that* in the string, which belongs to the same clause but occurs towards the end of the sentence. This results in large *item tenure*. While the parser holds the item in memory, it processes the DP which “intervenes” between *the* and *that* because the promotion structure derives stacked clauses using roll-up movement, as can be seen from the movement arrows in (34). And it is not difficult to tell that as the number of stacked clauses increases, *the* and *that* are separated by more materials,  $\text{MaxT}_{item}$  increases accordingly.

(34)



Next, we examine the adjunction structure schematized in (35).

(35)



The structure in (35) corresponds to the three-clause sentence in (33b). Two things are worth pointing out about the structure. First, (35) assumes a *wh*-movement analysis of relative clauses (Chomsky 1977). The head noun c-commands the relative pronouns *wh-rel* in each clause. They also stand in matching/agreement relation ensuring the correct interpretation of the relative pronouns (Aoun and Li 2003). Next, the multiple adjunction of CPs is one way to organize stacked relative clauses. It is chosen for simplicity because it involves no additional silent heads while maintaining binary branching.

Based on the current model, the processing difficulty associated with the adjunction structure is roughly consistent with the most complex clause, irrespective of the number of stacked clauses. This seems more aligned with our intuition. In the examples in (33) where the stacked clauses are equally complex,  $\text{MaxT}_{item}$  is 14 for both two-clause and three-clause sentences. For example, in the three-clause structure in (35),  $\text{MaxT}_{item}$  is found on the  $\text{CP}_i$  node. This node is stored in memory when the parser processes the word *cheese* and is removed from memory until the first relative clause  $\text{CP}_{ii}$  is built. A two-clause structure has the same  $\text{MaxT}_{item}$  profile, which readers can verify in Appendix A.3 Figure 9.

Returning to the three-clause structure in (35), as the first clause is built, the parser conjectures its sister node  $\text{CP}_{iii}$  and holds it in memory. This node remains in memory until its left-corner,  $\text{CP}_{iv}$ , is built. If  $\text{CP}_{iv}$  is sufficiently complex,  $\text{MaxT}_{item}$  could then shift to the  $\text{CP}_{iii}$  node. Finally, with the right structure,  $\text{MaxT}_{item}$  can also be trivially found in any of the stacked clauses.

It is beyond the scope of this paper to determine the correct syntactic structures for stacked relative clauses or to experimentally verify the processing difficulties that arise as the number of stacked clauses increases. Rather, the discussion is to highlight the potentials of our processing model as a tool for syntactic proposal verification, which future research can further exploit.

#### *Other Tenure-based metrics in LC MG parsing*

4.4

$\text{MaxT}_{item}$  is a tenure-based complexity metric that effectively captures the processing phenomena in our model. The calculation of  $\text{MaxT}_{item}$

relies solely on annotated derivation trees. This is easier and more intuitive for a human observer than delving into the trace of a parse.  $\text{SumT}_{item}$  and  $\text{AvgT}_{item}$  are two additional complexity metrics derivable from annotated derivation trees.

$\text{SumT}_{item}$ , the total non-trivial *item tenure* (i.e., tenure greater than 1) of the entire parse, is calculated by adding up the differences of each dash-connected indices that are greater than 1 and then dividing the total by 2. Dividing the sum of differences by 2 gives the correct  $\text{SumT}_{item}$  because only parse items that correspond to two nodes (implication items) can be stored for non-trivial steps. For such an item, its tenure is always reflected in the matching dash-connected indices on a pair of nodes, as discussed in Section 2.2.

$\text{SumT}_{item}$  for the current model can be found in Table 3.

Table 3:  
Modeling results  
based on  
 $\text{SumT}_{item}$

	$\text{SumT}_{item}$		
	Left	center	right
1-layer <sub>arc-eager</sub>	6	30	20
2-layer <sub>arc-eager</sub>	8	84	38

Overall, the 2-layer sentences have a higher  $\text{SumT}_{item}$  than their 1-layer counterparts. Since  $\text{SumT}_{item}$  measures memory usage over the entire parse, the results are expected as it reflects general factors such as length that impact processing. Moreover, from 1-layer to 2-layer sentences, the  $\text{SumT}_{item}$  of center-embeddings grows faster than that of left- and right-embeddings. This aligns with the trend that  $\text{MaxT}_{item}$  exhibits.

$\text{AvgT}_{item}$ , the average tenure of all tenured items, is also derivable from annotated derivation trees.  $\text{AvgT}_{item}$  equals  $\text{SumT}_{item}$  divided by the number of tenured parse items. And the number of tenured items is the number of unique dash-connected index pairs whose difference is non-trivial.

$\text{AvgT}_{item}$  for the current model can be found in Table 4. The trend found in the predictions of  $\text{SumT}_{item}$  continues in those of  $\text{AvgT}_{item}$ . As the number of layers increases,  $\text{AvgT}_{item}$  grows faster

Table 4:  
Modeling results  
based on  
 $\text{AvgT}_{item}$

	$\text{AvgT}_{item}$		
	Left	center	right
1-layer <sub>arc-eager</sub>	2	4.29	2.86
2-layer <sub>arc-eager</sub>	2	7	3.16

for center-embeddings than for left- and right embeddings. For the left-embeddings,  $\text{AvgT}_{item}$  is the same for both layer conditions because all tenured items happen to have a tenure of 2.

The above test-run of  $\text{SumT}_{item}$  and  $\text{AvgT}_{item}$  along with the main results from  $\text{MaxT}_{item}$  shows their potential as valid complexity metrics for LC MG parsing models. It is worth further exploring their empirical coverage as well as their interactions in potential ranked metrics discussed in Graf *et al.* (2017). We leave these to future research.

## CONCLUSION

5

To conclude, the modeling results suggest that left-corner parsing for MGs successfully captures human processing differences in left-, center-, and right-embeddings.  $\text{MaxT}_{item}$  is shown to be a valid complexity metric for our processing model. The results extend the parsing account for this processing contrast to another grammar formalism, MGs, and suggest that left-corner parsing for MGs is viable as a psycholinguistically adequate model for human sentence processing. The proposed tree annotation scheme invites future research into the space of proper complexity metrics for LC parsing for MGs.

## REFERENCES

- David ADGER (2003), *Core syntax: A minimalist approach*, volume 33, Oxford University Press Oxford.
- Joseph E. AOUN and Yen-hui Audrey LI (2003), *Essays on the Representational and Derivational Nature of Grammar: The Diversity of Wh-Constructions*, The MIT Press, ISBN 9780262267229, doi:10.7551/mitpress/2832.001.0001, <https://doi.org/10.7551/mitpress/2832.001.0001>.
- Yehoshua BAR-HILLEL (1966), Language and information; selected essays on their theory and application, *Foundations of Language*, 2(2):192–199.
- Noam CHOMSKY (1977), On wh-movement, *Formal Syntax*, pp. 71–132.
- Noam CHOMSKY (2014), *The minimalist program*, MIT press.

- Aniello DE SANTO (2019), Testing a minimalist grammar parser on italian relative clause asymmetries, in *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pp. 93–104.
- Thomas GRAF, James MONETTE, and Chong ZHANG (2017), Relative clauses as a benchmark for minimalist parsing, *Journal of Language Modelling*, 5(1):57–106.
- Tim HUNTER, Miloš STANOJEVIĆ, and Edward STABLER (2019), The active-filler strategy in a move-eager left-corner minimalist grammar parser, in *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pp. 1–10.
- Fred KARLSSON (2007), Constraints on multiple center-embedding of clauses, *Journal of Linguistics*, 43(2):365–392.
- Richard S KAYNE (1994), *The antisymmetry of syntax*, 25, mit Press.
- Gregory M KOBELE (2021), The saxon genitive, unpublished manuscript.
- Gregory M KOBELE, Sabrina GERTH, and John HALE (2013), Memory resource allocation in top-down minimalist parsing, in *Formal Grammar*, pp. 32–51, Springer.
- Susumu KUNO (1974), The position of relative clauses and conjunctions, *Linguistic Inquiry*, 5(1):117–136.
- So Young LEE (2018), A minimalist parsing account of attachment ambiguity in english and korean, *Journal of Cognitive Science*, 19(3):291–329.
- Lei LIU (2022), *Phrasal weight effect on word order*, Ph.D. thesis, State University of New York at Stony Brook.
- Philip RESNIK (1992), Left-corner parsing and psychological plausibility, in *COLING 1992 Volume 1: The 14th International Conference on Computational Linguistics*.
- Stuart M SHIEBER (1985), Evidence against the context-freeness of natural language, in *Philosophy, Language, and Artificial Intelligence*, pp. 79–89, Springer.
- Edward STABLER (1997), Derivational minimalism, in *Logical Aspects of Computational Linguistics: First International Conference, LACL'96, Nancy, France, September 23-25, 1996. Selected Papers*, volume 1328, p. 68, Springer Science & Business Media.
- Edward P STABLER (2011), Computational perspectives on minimalism, *Oxford handbook of linguistic minimalism*, pp. 617–643.
- Miloš STANOJEVIĆ and Edward STABLER (2018), A sound and complete left-corner parsing for minimalist grammars, in *Proceedings of the Eight Workshop on Cognitive Aspects of Computational Language Learning and Processing*, pp. 65–74.
- Laurie A STOWE (1986), Parsing wh-constructions: Evidence for on-line gap location, *Language and cognitive processes*, 1(3):227–245.



*Psycholinguistic adequacy of LC parsing for MGs*

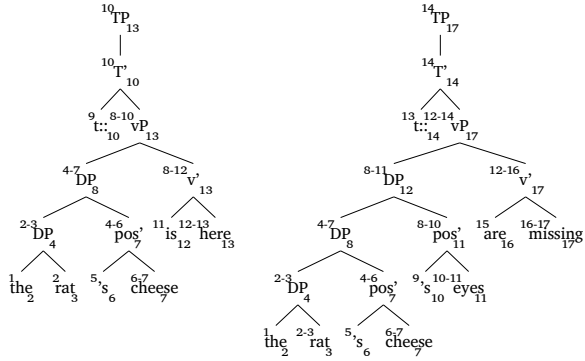
Chong ZHANG (2017), *Stacked relatives: Their structure, processing and computation*, Ph.D. thesis, State University of New York at Stony Brook.

A

APPENDIX

A.1 *Tree annotations for arc-standard LC MG parsing*

Figure 4:  
Tree annotations  
for arc-standard  
left-embeddings



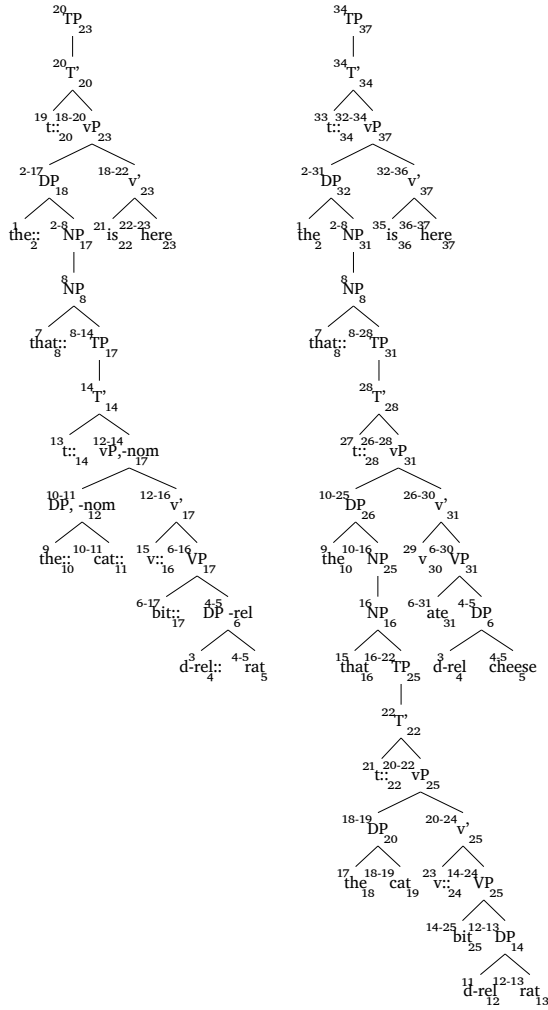


Figure 5:  
Tree annotations  
for arc-standard  
center-  
embeddings



Movement-based analysis for left-embeddings

A.2

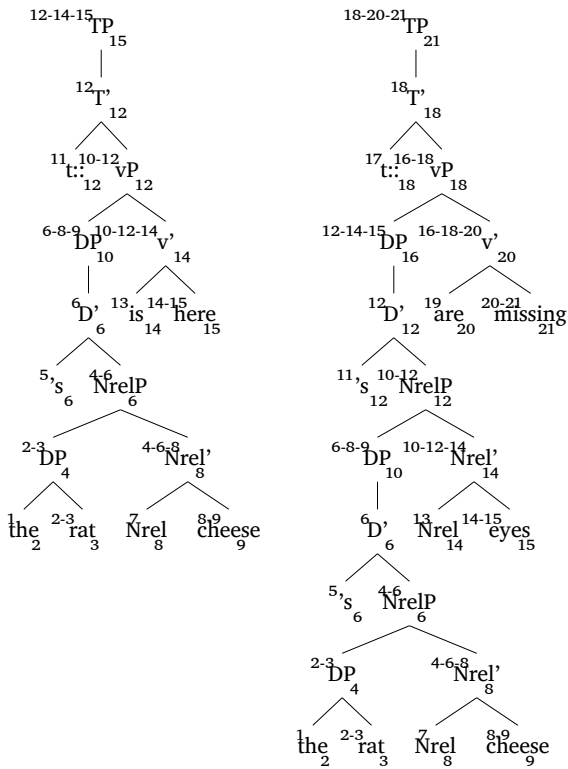


Figure 7:  
Tree annotations  
for  
movement-based  
left-embeddings



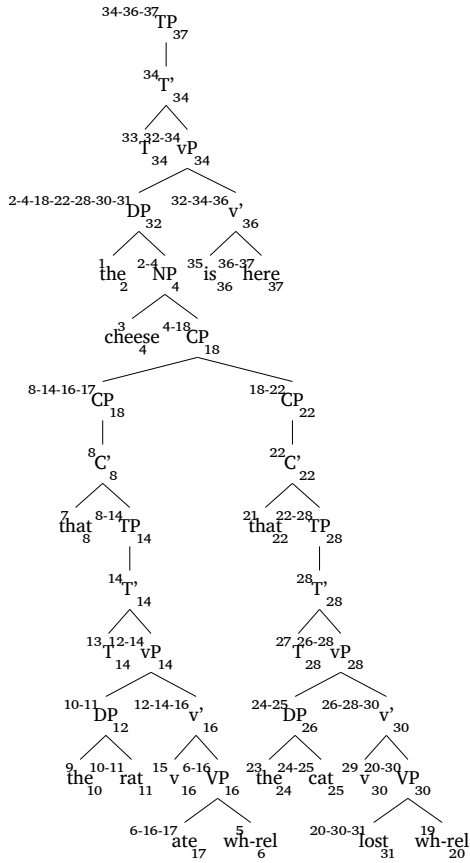



Figure 9:  
Tree annotations  
for stacked  
relative clauses  
(*wh*-movement  
analysis)

*Anonymous*

This work is licensed under the *Creative Commons Attribution 4.0 Public License*.

 <http://creativecommons.org/licenses/by/4.0/>