The Role of Working Memory in Syntactic Sentence Realization: A modeling & simulation approach

Jeremy R. Cole and David Reitter
The Pennsylvania State University
jrcole,reitter@psu.edu

Abstract

This paper examines the effects of working memory size in incremental grammatical encoding during language production. Our experiment tests different variants of a computational-cognitive model that combines an empirically validated framework of general cognition, ACT-R, with a linguistic theory, Combinatory Categorial Grammar. The model is induced from a corpus of spoken dialogue. This methodology facilitates comparison of different strategies and working memory capacities according to the similarity of the model's produced sentences to the corpus sentences. The experiment presented shows that while having more working memory available improves performance, using less working memory during realization does as well, even after controlling sentence length. Sentences realized with a more incremental strategy also appear to more closely track the naturalistic data. As high incrementality is correlated with low working memory usage, this study offers a possible mechanism by which syntactic incrementality can be explained. Finally, this paper proposes a multi-disciplinary modeling and simulation-based approach to empirical psycholinguistic inquiry.

Keywords: cognitive modeling, working memory, language production

1. Introduction

Working memory has long been thought to play an important role in language processing (e.g., Gibson, 1998). However, the precise role of working memory in *grammatical encoding*, the process by which words are combined into sentences, remains unclear. This paper compares different computationally implemented models of grammatical encoding by fitting them to empirical data. Such an approach can reveal how well a model fits the data when available working memory and grammatical encoding strategies are varied.

Many unanswered questions about grammatical encoding concern planning. For instance, how *incremental* is the planning process? Are words and phrases planned in the exact order that they are output, or are some of them planned earlier and kept in some form of buffer? There are substantial opportunities to leverage representational insights from computational linguistics to answer such questions.

Moreover, linguistic models such as grammar formalisms do not specify a memory component (e.g., Steedman, 2000a). This makes it difficult to make specific claims about processes related to general cognition, such as buffering. Conversely, connectionist models often aim to be more agnostic to the linguistic representations of the task (e.g., Dell et al., 1999), making them challenging to interpret once they are trained on data. In short, the general motivations to combine cognitive modeling and computational linguistics apply to grammatical encoding.

With the availability of large-scale data, language models should strive to explain as much data as possible. Past psycholinguistic models have managed complexity by focusing on specific, empirically observable effects and syntactic construction. For example, accounts of syntactic priming explain how speakers re-use syntactic alternatives, such as a double object ("She gave John the ball") and prepositional object ("She gave the ball to John") constructions (e.g., Pickering and Branigan, 1998). Cognitive models broadened which syntactic phenomena are covered using connectionist frameworks (e.g., Chang et al., 2006), or as higher-level ACT-R models (Reitter et al., 2011). While these models are capable of learning an arbitrary number of sentences, they are primarily concerned mainly with a number of low-frequency syntactic constructions. Thus, computationally modeling language production in a plausible way remains a challenge.

Big data, used appropriately, allow us to cover arbitrary syntactic constructions. Additionally, we can contrast different model variants in terms of their explanatory power. This methodology can lead to incremental improvements. In short, we are pursuing *big data cognitive modeling*: instead of modeling a relatively small number of experiments surrounding a phenomenon, we model a large amount of the raw data produced by the phenomenon itself. So, for the task at hand, we evaluate against a corpus.

The methodological approach taken in this paper is one of *modeling & simulation*. The models are based on carefully chosen combination of grammatical theory and a computational psychological account of human cognition. The implemented model that we will discuss has clear, interpretable representations in the form of Combinatory Categorial Grammar (CCG; Steedman and Baldridge, 2011). It is cognitively plausible and implemented in an empirically validated framework, ACT-R (Anderson et al., 2004). It is empirically testable, as the model can produce output for any target sentence, allowing competition among alternative models. The motivation to model and simulate arises from our desire to explore the consequences of theories for working memory usage, and their interplay with flexible incrementality. We ground our simulation in large-scale empirical data of human speech in order to be able to evaluate different modeling hypotheses.

Our experiment in particular examines *incrementality* in grammatical encoding, working memory *availability*, and actual *working memory usage* and how these variables are associated with an improved or worsened fit of the model to linguistic data.

2. Assumptions and Contribution

This paper proposes a model of language production that relies on the integration of formal grammar and an empirically-verified theory of memory. Some readers may be skeptical whether all aspects of the model reflect the human language production process. The goal of this paper is not to suggest that the presented model uses precisely the same representations and algorithms as the human language production process. To examine the hypothetical question of what would happen if verbal working memory was more or less limited, a computationally implemented, computationally evaluable simulation is necessary. The representations that we use in the model allow for this interpretation.

Some of the psycholinguistic assumptions made here are of consequence for the outcome of the simulation. Notably, CCG's representation is plausible in the memory required for grammatical encoding, as our process does not require active manipulation of the tree structure or actively exploring alternatives at realization time, which may be psycholinguistically implausible based on limited memory. Our model assumes that language processing depends on a series of declarative memory retrievals and the invocation of proceduralized routines, as constrained by ACT-R. We do *not* assume that verbal working memory or procedural knowledge occupy precisely the same cognitive resources as general working memory or procedures. Likewise, we do not assume that internal representations of syntax necessarily look exactly like CCG. Thus, evaluating whether or not our particular model is the most accurate possible model is outside the scope of the work.

To reiterate, our goal is to make a model that is inspectable along both cognitive and linguistic axes, rather than to make the highest performing model we can imagine.

3. Related Work

This paper builds on and attempts to unify previous models of grammatical encoding and memory by using a methodology based on modeling and unsupervised evaluation on corpus data.

3.1. Grammatical Encoding

Grammatical encoding refers to a stage of the language production process. In this sense, we say that after an idea is formulated, it is grammatically encoded and then phonologically encoded (Bock and Levelt, 2002). In turn, grammatical encoding consists of lexical selection, function assignment, and constituent assembly. The first stage maps ideas to words; the second stage maps words to parts of speech, and the last stage combines these lexical-syntactic units (hereafter lexsyns) into constituents. As syntactic trees are formed by recursively combining constituents, this process eventually leads to a sentence. Thus, the full process of grammatical encoding transforms ideas into sentences. We do not claim that this is the only useful discretization of the language production process; however, by focusing in particular on grammatical encoding, our model avoids some of the messiness surrounding representing ideas, which is a difficult task in a model with the desired level of interpretability.

These exact stages were chosen for their usefulness in explaining some empirical data, such as speech errors. However, we do not claim these exact stages are necessary in every useful process model. The presented model allows for the free combination of compatible lexical-syntactic elements, focusing on the stages of function assignment and constituent assembly, in order to investigate questions about memory usage and incrementality.

3.2. Working Memory

The literature proposes a rich variety of constraints on sentence formulation as a result of working memory (WM) available. For instance, a higher

WM span can decrease certain types of grammatical errors (Hartsuiker and Barkhuysen, 2006, Badecker and Kuminiak, 2007). Further, Slevc (2011) suggests working memory load can affect the incrementality of a sentence. Importantly, the (verbal) working memory we refer to is not necessarily the same as general working memory (see e.g., Baddeley, 1992, Acheson and MacDonald, 2009).

Nonetheless, these studies make no explicit claims as to the representation of language in memory during the function assignment and constituent assembly process, and obviously such a representation will interact with how WM is used. Instead, representational questions in syntax have remained the domain of linguistic theory. Here, we follow Combinatory Categorial Grammar (Steedman and Baldridge, 2011), which provides a set of possible actions and associated temporary representations during grammatical encoding. The proposed model maps these representations to the psychological architecture of attention, processing and memory.

3.3. Incrementality in Language Production

Our model's evaluation is based on exploring incremental planning in the production process and the planning process's relationship with working memory. In grammatical encoding, *incrementality* refers to when and in which order syntactic choices are made. For instance, all of the choices could be made before phonological processing starts, which would be fully non-incremental. Conversely, they could be made immediately before a word is uttered, which would be completely incremental.

In language comprehension, material becomes available incrementally, thus incremental processing is assumed. However, this is not the case in language production: it is unknown how incrementally encodable semantic material becomes available. This process is nonetheless governed by limitations on working memory: buffers are insufficient to plausibly retain the entire structure of longer sentences based on the present understanding of working memory (Baddeley, 2003). Previous models have nonetheless assumed a mostly incremental strategy (Bock and Levelt, 2002, Guhe, 2007), while we aim to systematically investigate the effect of varied working memory capacity.

Ferreira (1996) makes an argument for incrementality, based on the observation that competing syntactic alternatives facilitate production rather than inhibit it. An incremental account of sentence realization would predict such an effect, as the alternatives makes it easier to find a workable syntactic decision. By contrast, in a non-incremental account, competing material would slow down the process as it leads to combinatory explosion. Further results, however, relativize this account of incrementality when it comes to the syntax-phonology interface (Ferreira and Swets, 2002). Incremental production is possible, but it is "under strategic control"; it depends on semantic information, and it could be modulated by external factors, such as stress.

Thus we assume that incrementality in grammatical encoding is graded: the degree to which a sentence is realized incrementally may vary based on certain cognitive factors. Nonetheless, previous work does not integrate this with an account of memory. Our corpus-driven method works to explain this by contrasting models with different available working memory, and by examining actual working memory use, while calculating the activation and availability of linguistic structures.

3.4. Cognitive Models of Language

ACT-R is a general theory of cognition (Anderson et al., 2004). ACT-R, combined with a linguistic theory like CCG, can provide a unification of computational modeling, cognitive science, and linguistics. ACT-R's basic system for writing models involves chunks and production rules, where chunks represent declarative memory and production rules represent procedural memory. In the later sections, we describe our particular model in more detail.

Cognitive modeling within a cognitive architecture such as ACT-R (Anderson et al., 2004) has advantages to explicit hypothesis testing. For instance, ACT-R already specifies a general theory of memory, which has been empirically validated on a variety of tasks. The symbolic nature of ACT-R lends to the interpretability of the final, produced model. Models in ACT-R provide a computable, unified account of the process in question, from defined start points to defined and testable end points. Further, they have theoretical integration with a wide variety of tasks, including performance measures such as reaction times or neurophysiological effects.

There is also some history in using cognitive architectures to combine theories of memory and linguistic representations, primarily in language comprehension. For instance, Van Rijn and Anderson (2003) explained the lexical decision task as a by-product of chunk activation. Further, both Lewis and Vasishth (2005) and Ball et al. (2007) make strides toward more general models of language comprehension with the usage of a linguistic theory.

There are fewer ACT-R models that carefully integrate language *production* and linguistic theory (see Vogelzang et al., 2017, for a review). Such models are not yet evaluated with respect to the breadth of syntactic alternatives available to a speaker, even if they show the possibility of combining linguistic representations and a model of memory to answer some psycholinguistic questions.

3.5. Combinatory Categorial Grammar as a Psycholinguistic Theory

Combinatory Categorial Grammar is a grammar formalism (Steedman and Baldridge, 2011). Our model uses both its representations and combinatory mechanism; the important psycholinguistic consequence of this is that CCG simplifies its representation after each syntactic operation. This is as opposed to other grammar formalisms, where combination traditionally results in a more complicated representation, e.g. Tree-Adjoining Grammar (Joshi and Schabes, 1997). In general, grammar formalisms operate based on *types*, such as noun phrase, and *rules*, which are methods for combining the types into a sentence. See Table 1 for a demonstration of CCG's combinatory rules. See Figure 1 for an example of how these rules can create sentences.

CCG can also produce multiple different parses for a single sentence, which could correspond to a speaker varying their strategy depending on the context. This is done through *type-raising*, a special rule where a type changes to an equivalent type that can take arguments. An example of type-raising can be seen in Figure 1. Fundamentally, type-raising changes something from an *argument* to a *function*. Intuitively, this means that instead of choosing noun phrases to match a verb, a speaker would choose a verb to match their noun phrase.

Because of this ability to produce multiple parses and its representational compactness, grammatical encoding in CCG can be done with constant (O(1)) space and linear (O(n)) time. No sentence ever requires more than N-1 combinatory rule applications. These affordances allow us to create a cognitively plausible model with CCG, that does not require cognitive resources that do not have any empirical justification.

4. Research Questions

Our model has the potential to answer several important questions. As the model is naive and makes few theoretical assumptions, the strategy it uses to realize sentences can vary from sentence to sentence. This allows

Table 1: LHS: the two types to be combined. RHS: resultant type after the operation. There are a limited number of *atomic* types (e.g., *NP*); most types are recursively specified (e.g., (S/NP) \ NP). All of these types, whether recursively specified or atomic, could potentially take on the roles of X, Y, or Z in any of the rules. For instance, an example forward applications with two recursive input types could look like: (NP\N)/(S/NP) > (S/NP) = (S/NP), or with two atomic types: S/NP > NP = S.

Forward Application (>):	$X/Y > Y = X$
Backward Application (<):	$Y < X \setminus Y = X$
Forward Composition (>>):	$X/Y >> Y/Z = X/Z$
Backward Composition (<<):	$Y \setminus Z << X \setminus Y = X \setminus Z$

us to see which kind of strategies result in output that is ultimately similar to what humans actually produced. In particular, we are interested in the actual syntax trees the model uses. These syntax trees give us a very high fidelity trace of the model's strategy that is near-unobtainable in human experiments. These syntax trees can thus provide us additional evidence in debates over syntactic strategies and memory usage in language production.

Ferreira and Swets (2002) suggested that while under heavier stress, participants relied on more incremental constructions. In their study, they guided participants to respond incrementally or non-incrementally by providing templates. The choice of materials seems important, as it might well influence the efficiency with which more or less incremental strategies can be applied. A corpus evaluation provides a wider array of materials, at the price of leaving control over the encoding strategy to the speakers. We can, however, contrast generative models that use more or less incremental strategies in different situations, or that have different cognitive resources available. For example, if incremental processing is the generic strategy for language production, we would expect the model to have higher performance when it is more incremental.

Meanwhile, Hartsuiker and Barkhuysen (2006), Badecker and Kuminiak (2007), and Slevc (2011) provide converging evidence that a lower working memory capacity should make production errors more common. However, examining participants with different working memory spans does not control for other possible factors that may correlate with working memory spans. Moreover, occupying participants' working memory could cause interference effects due to a linguistic encoding,. A model could provide additional evidence toward answering this question, as the working memory can be directly manipulated. Additionally, these studies do not tell us how people use working memory during language production. A cognitive model can provide such a trace, and we can assume that a higher performing model is providing a more accurate trace.

5. Model

In the following sections, we describe how a model is derived from the syntactic and lexical information present in $1,200$ sentences sampled from

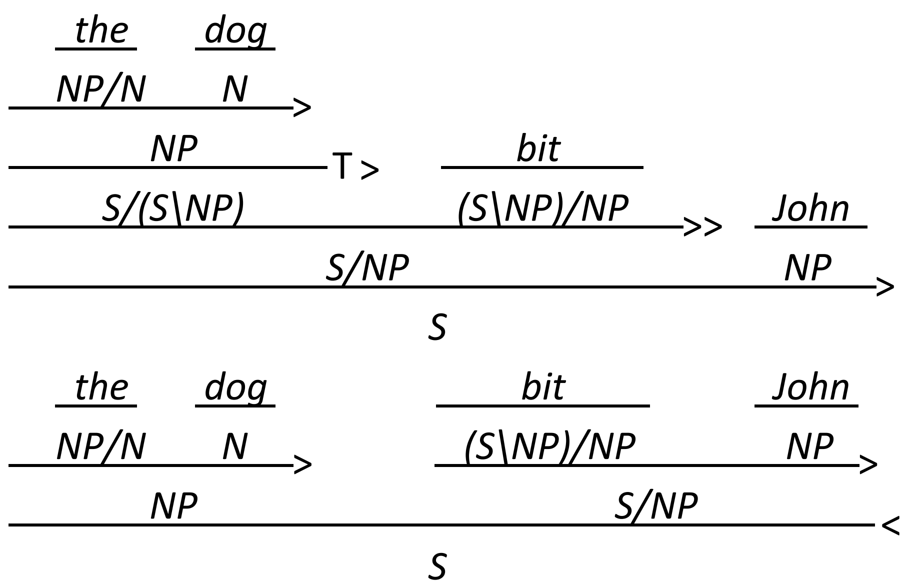

Figure 1: Two contrasting CCG derivations: The top is more incremental (right-branching) than the bottom. Note that type-raising is normally used in non-standard derivations, which are more incremental rather than following the normal-form phrase-based derivation. In general, purely by CCG, every sentence could be parsed incrementally or non-incrementally, but taking into account other cognitive mechanisms, such as memory activation, may make some sentences easier to parse incrementally than others. The parse trees should be read from top to bottom, where two rule applications at the same height could have occurred in either order. On each line is the surface form of a word on top and its type on bottom (with no rule symbol at the end of the line), or two input types on top with the resulting type on bottom (with the applicable rule symbol at the end of the line).

the *Switchboard* corpus (Godfrey et al., 1992) [1]. We then run this model with no interruptions or constraints, using the unordered bag-of-words from the corpus sentences as input, expecting sentences or sentence fragments as output. The model's process is recorded in the form of syntax trees ("derivations") for further analysis, as these derivations reflect the strategy applied by the model to produce the sentences. The model's performance under different working memory conditions will be evaluated by comparing to each original sentence. Thus, the core task of the model is to recover the original ordering of the words in each sentence.

5.1. Model

The model, fundamentally, is an integration of ACT-R (Anderson et al., 2004) with CCG (Steedman, 2000b). CCG specifies clear symbolic and procedural components that map naturally to chunks and production rules. As discussed earlier, using CCG as the combinatory mechanism of the cognitive model means that combinations will reduce the current memory use, which we see as consistent with current ideas about chunking (Conway et al., 2005).

The model forms a sentence by greedily combining lexical-syntactic chunks together. It treats no words, types, or rules as special, and its only knowledge of how words combine is derived directly from the constraints of CCG. However, simply following CCG rules can lead to unidiomatic sentences and even ungrammatical sentences by violating thematic constraints. This is true of the presented model. Nonetheless, the full expressive range of syntactic and lexical constructions found in a corpus would require some form of statistical learning mechanism: we consider this out of scope. Indeed, our model's goal is not to produce statistically useful output but to carefully examine the processes that control the output. Randomly selected example sentences produced by the model can be found in Table 2.

5.1.1. Declarative Memory

Declarative Memory (DM) is composed of items which are called *chunks*. In our model, these chunks have four different types: Sentence, Lexsyn, Word, and Type. Chunks have some number of *slots* which store the data about that chunk. Different types of chunks have different number and type

[1] For machine learning models, 1200 sentences is not big data; however, it is much more than cognitive models are normally evaluated on, and the methodology could be expanded further

of slots. Slots themselves also contain chunks, though at any given time they can be empty. Thus, chunks are defined somewhat recursively. The simplest chunk has no slots, so the only data it contains is its name.

The chunks in our model are organized such that Sentences are the most complicated type, containing some number of Lexsyns in their slots, while Lexsyns are composed of a single Word and several Types which correspond to the variety of function assignments the word could take on in various contexts. The organizational scheme of the chunks can be found in Figure 3.

Words simply have a name (i.e., no slots), which corresponds to the word's surface form (e.g., *family*). Words are pulled directly from the corpus.

Types are arbitrarily complex CCG types. The types that exist in DM are the types that are used in Switchboard CCG derivations of our chosen sentences (e.g., Noun Phrase). The type's slots include a left part, right part, and an operator connecting the two parts. For instance, the type S/NP would have slots for S, /, NP. This means that the result of a syntax rule application is immediately retrievable. Thus, the types are specified recursively from some number of base types that exist in the corpus.

Lexsyns associate a Word with some number of Types. These associated types are taken from the function assignments of each word in the Switchboard CCG derivations of our chosen sentences. The types are ordered from most common to least common, which would mean more common types would be selected if multiple options are available and the utilities have not yet changed.

Sentences are the goals of the model and are equivalent to the corpus sentences. The model retrieves a sentence it wishes to encode, which is initially represented as a bag of words. The Sentence chunk also contains the current state of grammatical encoding as slots for some number of Lexsyns, which becomes the model's concept of Working Memory (WM). As a reminder, at any given point, some of the slots could be empty, which would correspond to reduce working memory usage. Thus, varying the number of slots for lexsyns corresponds to varying the working memory capacity of the model. The Input and WM existing in a single chunk is not a theoretical commitment; however, due to our focus on grammatical encoding, we had to assume the previous tasks of idea generation and lexical selection were complete. In reality, it is likely that all three tasks overlap to some extent. Thus, in the following descriptions, we refer to WM and Input as separate components.

Table 2: 'Target': actual corpus sentences vs. model's realizations. These sentences were selected randomly. They feature realizations that are somewhat unidiomatic and at times arguably ungrammatical.

Realization	Target
downhill going like everybody	but then they started going downhill like everybody else
you fire never something unless anybody 're caught	they never fire anybody unless you 're caught doing something illegally
still taxes raise probably and	i think he can probably raise taxes and still get elected
i then and decided i like author this	and then i decided i like this author
are school working you	are you working anywhere while you are going to school

5.1.2. Production Rules

Production rules in ACT-R are defined with pre-conditions and effects. In our case, with the grammar defining the production rules, these production rules have similar pre-conditions and effects as their corresponding syntax rules.

We define a small set of about ten production rules, which are compiled into several thousand production rules through an automatic process. This process essentially expands these production rules into rules that are theoretically similar, but have a slightly form because of the limitations of ACT-R's symbolic and ordinal production system. For instance, combining two lexsyns by a single syntax rule will require production rules based on which type variant the lexsyn is combining under.

The architecture will choose a production rule based on the current state and that rule's constraints; there is no predefined algorithmic flow. If multiple rules are available, the one with the highest utility will be selected. Utility learning is one of ACT-R's integrated learning paradigms for procedural learning (Taatgen et al., 2006). The basic architecture of the model's production rules can be found in Figure 2. The model's production rules fall into a few basic categories.

Syntax Rule Applications occur when WM contains at least two Lexsyns whose types would follow the constraints of at least one CCG rule.

For instance, if Working Memory contains Family (NP/NP) and Home (NP), then it could use Forward Application to combine them into a single chunk, Family Home (NP). The result is found with a Type Retrieval.

Adding Word From Input to Working Memory can occur whenever there is free space in WM. It removes the word from the Input and initiates a Type Retrieval for its function assignment.

Flushing can occur when no other rules apply. The model removes something from WM to try again. From a cognitive perspective, this could be thought of as backtracking.

Type Retrieval generally occurs to clean up after a previous rule called for a type to be retrieved from Declarative Memory. It retrieves word's function assignments or the result of a syntax rule.

5.2. Model Generation

The model itself is induced from a sample of Switchboard. Our sample is obtained from sentences up to fifteen words. Additionally, a filter is applied that limits the sentences' 'syntactic complexity' by limiting infrequent function assignments. The practical use of this filtering is to limit the total number of type assignments that are possible for every given word in the model. After that, 1,200 sentences were selected at random.

These sentences provide a lexicon of words, a mapping of words to types, and a list of target sentences. The model stores these as Words, Lexsyns, and Sentences respectively. The model's production rules are then generated procedurally: every type of syntax rule, such as forward application, specifies individual rules for type frequency and working memory slot. To reiterate, the model learns nothing about the ordering of the words from the corpus; it simply learns their possible function assignments.

5.3. Creating Syntax Trees

In realizing a sentence, the model creates one or more sentence fragments. Since the exact order that the model combines two lexsyns is readily inspectable, deriving the syntax tree is fairly simple. When two lexsyns are combined, their nodes on the syntax tree are combined. If the realization is incomplete, then there would be a syntax tree for every fragment that was partially realized, though single element trees were discarded.

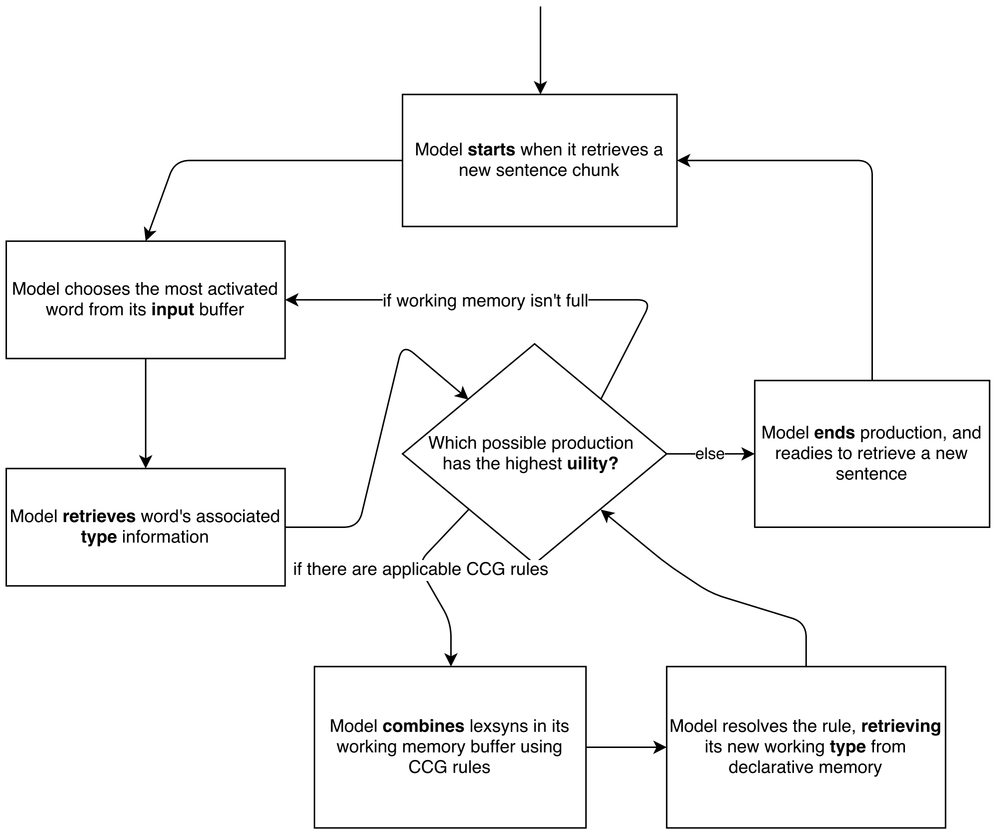

Figure 2: Process flow of the production rules of the model. Productions use the constraint matching mechanism of ACT-R.

5.4. Integration of ACT-R and CCG

The version of ACT-R we use is essentially canonical. It has a fixed declarative memory and learns nothing besides utilities. Utilities are a simple learning mechanism that affect which rule will be selected if multiple different rules match based on previous success with that rule. In other words, the model does not learn anything inherent about the ordering of words; it only knows their syntactic relationships, and it can make decisions based on utility about which syntactic relationships might be more appropriate to apply based on utility. Moreover, when choosing whether to process more semantic information (thus increasing the amount of working memory currently in use) or to contract the current representation with a syntactic

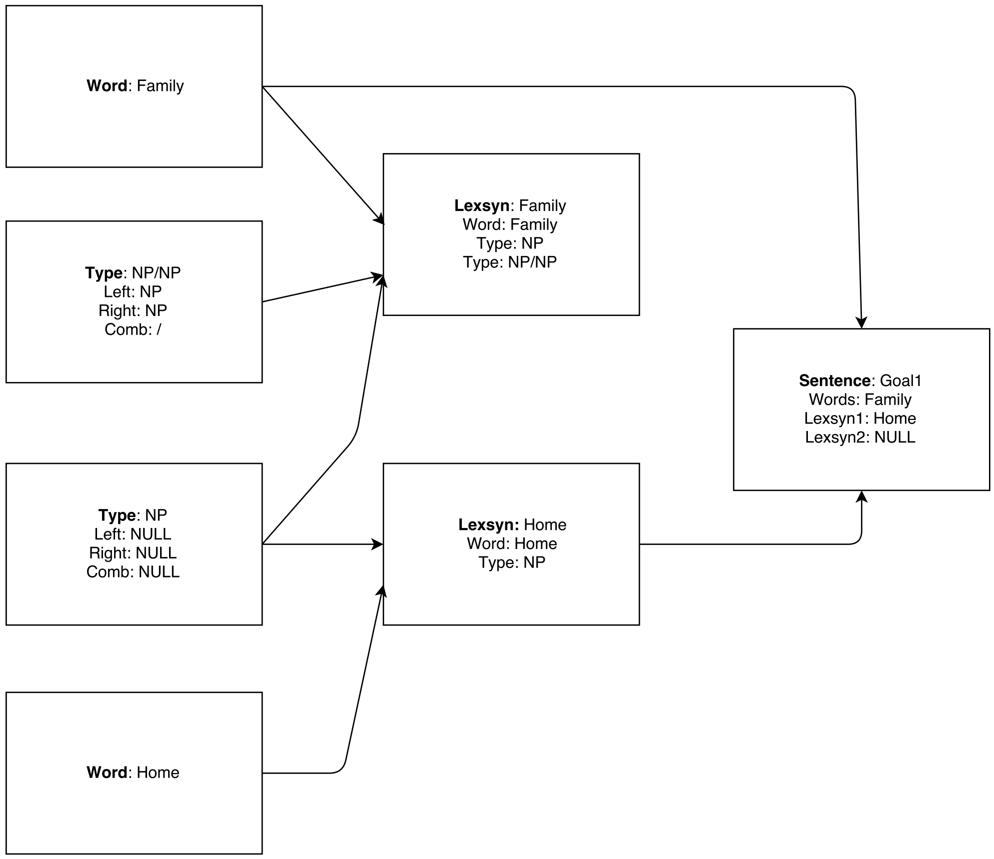

Figure 3: Data model of declarative memory. Arrows indicate "has-a" relationships. Example: vocabulary {Home, Family} and types {NP/NP,NP}; the phrase to be produced is *Family Home*; so far, *Home* and that word's type has been added to WM. The next step will be to retrieve *Family*, and then combine them with Forward Application.

rule application, the model can also rely on utility. Fundamentally, however, the model makes decision's based on its hardcoded understanding of CCG rules and types.

The CCG that is hardcoded into the model is slightly different than canonical CCG. In particular, while the basic combinatory grammar rules exist as production rules, *type-raising* was removed. Type-raising is somewhat problematic for integration in ACT-R, since it would require the model to modify the function assignment of each chunk without yet knowing if the new function assignment will be useful. In its place, incremental constructions are handled by encoding type-raised types as possible function assignments of the chunks. Then, the model can vary its incrementality by choosing the type-raised type instead of the normal type. This can be done simply by having separate production rules for types and their type-raised equivalents, so the one that matches will be selected with no additional operation.

6. Simulation

As previously discussed, our interest is in how the incrementality of sentences is modulated by working memory. We look at the *quality* of the model's output under different conditions, as measured by its fidelity in achieving the target sentences. Moreover, we examine the *strategy* the model is using in each of these conditions to determine when its achieving the highest-quality results.

6.1. Variants

We contrast two versions of the model with different amounts of *working memory*. Our two conditions use three (WM3) and five (WM5) working memory slots, respectively. We consider these values as realistic lower and upper bounds of working memory capacity as found in language tasks (Daneman and Carpenter, 1980). This is implemented simply by limiting the number of slots in the Sentence chunk, so the model has less available working memory to use to combine Lexsyns. We distinguish working memory span (controlled) from actual working memory usage (observed).

6.2. Dependent Variables

Branching Factor. We see grammatical encoding as a flexible process: the set of production rules, and the absence of a fixed algorithmic process is commensurate with that, as is the ACT-R cognitive architecture in general. Strategies emerge as a result of the available cognitive resources, such as WM, and, ultimately (not modeled) the success of rule sets. We measure an important aspect of the strategy: incrementality, as determined by branching factor: The more right-branching a syntax tree is, the more incrementally it was realized.

We define two basic metrics for measuring branching factor. The Unweighted Branching Factor (UBF) is the number of right-branching decisions compared to the number of total decisions. The Weighted Branching Factor (WBF) takes into account how far up the syntax tree the decision was made;

it short, it sums all of the subtrees rather than simply comparing the decisions. An example tree and computation can be found in Figure 4, which is an syntax tree created from the model's syntactic decisions. Alternatively, to reference the CCG derivations from earlier in Figure 1, the top derivation has a WBF of 3 and a UBF of 7, while the bottom derivation has a WBF of 1.0 and a UBF of 1.0. These values are not on the same scale: 1.0 is the mean for WBF, but 0.2 is the mean of UBF. Both metrics correlate with each other and higher values represent more incremental constructions. The mean for WBF is very close to the suggested normal form derivation (i.e., the canonically accepted derivation by linguistics), so we consider it the stronger metric.

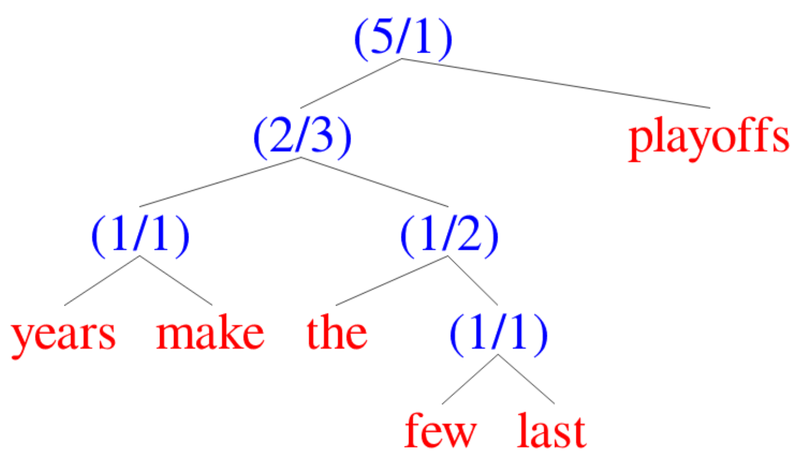

Figure 4: An example syntax tree built from actual model output. As can be seen, the model's output is frequently unidiomatic, even when it follows grammatical rules. This tree is created by following the actual derivation the model went through in order to produce the sentence, but without considering the actual syntax rules or types. Unlike the CCG trees, it should be read from bottom to top, where at a given height, the decisions could have been made in either order. To use this tree to compute WBF, sum the left numbers (corresponding to the number of nodes in the left subtree) and divide by the sum of the right numbers (corresponding to the number of nodes in the right subtree). Thus, WBF=1.25. To compute UBF, simply observe the number of left leaf nodes and divide by the number of right leaf nodes. Thus, UBF=1.0. For a fully right-branching or incremental tree, for instance, there could be only a single left leaf node (the inverse is also possible).

Working Memory Usage. Measured Working Memory Usage (MWMU) is a metric we define as the maximum amount of slots in WM the model used while realizing sentence q, including a separate slot for retrieval. So, for every state x_i, where k is the number of states during the grammatical encoding process ($0 \leq i \leq k$), where s_{ji} refers to whether slot j contains a chunk at state x_i (1 if contains a chunk, 0 otherwise), and n refers to the number of slots in WM ($0 \leq j \leq n$), and s_{ri} refers to the retrieval slot at state x_i:

$$f(x_i) = s_{1i} + ... + s_{ni} + s_{ri}$$

$$\text{MWMU}(q) = \max(f(x_1), ..., f(x_k))$$

For instance, in the perfectly incremental case, this would be two: each chunk is retrieved and appended rightward until the sentence is complete. In the perfectly non-incremental case, this would be the length of the sentence, though it would still be limited to the size of working memory. Thus, in general, the minimum value for MWMU is 2, and the maximum value is $n + 1$. Lastly, we also compute a metric that takes into account the length of the sentence, Adjusted Working Memory Usage (AWMU):

$$\text{AWMU} = \text{MWMU}/n$$

This metric is useful as longer sentences could possibly require additional working memory, especially if the constructions tend to be less incremental, so it serves as a control for sentence complexity.

Edit Distance. This measure evaluates fidelity of the model output, i.e., the match between the result and the input sentence (Levenshtein, 1966). An edit distance counts the number of changes (additions, swaps, and deletions of words) to transform one sentence into another one. If the model produces multiple fragments rather than a single utterance, the distances are averaged.

The measures correlates well with metrics used to evaluate natural language processing tasks such as ROUGE (Lin, 2004). We decided not to use ROUGE itself because its designed for semantic similarity; in particular, it was designed for the auto-summarization task. Thus, we did not think it was suitable for a task based on syntactic choices. Thus we determine our *model fit* as the average edit distance between the model's sentence and the target sentence.

7. Results

We examined the correlations between the branching factor, working memory usage, and fit, as measured by edit distance between the realization and the target sentences. We analyzed the influence of observed branching factor, available WM and observed WM usage separately.

Table 3: Individual Paired t-tests between WM=3 and WM=5 on each of the predictors for edit distance, along with a paired t-test for edit distance. This statistical test showed a significant reduction in edit distance for WM5, so it more closely fit the data.

	WBF	UBF	WMU	AWMU	dist
WM5-Mean	1.050	0.132	3.156	1.721	0.743
WM3-Mean	1.023	0.105	2.703	1.575	0.748
p-value	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001

Table 4: Four single-predictor linear models correlating each predictor with edit distance in the WM3 condition.

	WBF	UBF	WMU	AWMU
p-value	< 0.0001	< 0.0001	0.007	< 0.0001
effect	-0.221	-0.168	0.057	0.074
Intercept	0.873	0.763	0.507	0.610
r^2	0.056	0.025	0.162	0.065
df	1038	1038	1038	1038

Table 5: Four single-predictor linear models correlating each predictor with edit distance in the WM5 condition.

	WBF	UBF	WMU	AWMU
p-value	< 0.0001	< 0.0001	< 0.0001	< 0.0001
effect	-0.162	-0.228	0.120	0.041
Intercept	0.915	0.773	0.427	0.685
r^2	0.079	0.536	0.092	0.015
df	1038	1038	1038	1038

Both branching factor metrics (UBF and WBF) were found to be significant with a negative effect on edit distance, implying more incremental constructions produce realizations more similar to the initial sentences

($p < 0.001$). Conversely, both working memory usage metrics (MWMU and AWMU) were found to have a positive effect, implying increased WM usage decreased fit ($p < 0.001$). This had an even larger effect for AWMU, implying minimizing WM usage was especially important for longer sentences. Branching factor and working memory usage (all metrics) were also significantly correlated with each other ($p < 0.001$).

Because increased working memory usage is correlated with decreased fit, this begs the question of whether that is because sentences with lower working memory usage requirements are easier, or whether using more working memory directly decreases fit. As sentences have different working memory requirements, the ones with lower requirements could just be easier to realize incrementally, possibly reducing production errors. The five-slot model helps elucidate this.

The WM5 variant has significantly lower edit distance than the WM3 variant, so the WM5 variant of the model more closely fits the data. This effect is despite that it uses more working memory on average by both metrics. However, it is also more right-branching than the other model by both metrics.

8. Discussion

The final results of the simulation are nuanced: reducing working memory usage reduces fit, while increasing working memory capacity increases fit. Moreover, more right-branching constructions are uniformly associated with better fit.

8.1. Working Memory Usage

Increased working memory usage being associated with worsened fit is perhaps our most surprising finding. Previous studies have tried to artificially control the amount of working memory available to participants, but they were unable to measure the actual amount of working memory used. This finding is likely due to the model adopting a poor and possibly inhuman strategy on those sentences: proceeding breadth first and avoiding early commitment to syntactic choices. Ultimately, we would expect such a strategy to be unsuccessful, since it's at odds with empirical evidence of incremental commitment (e.g., Ferreira and Bailey, 2004).

Still, this association may seem counter-intuitive: dedicating more resources to grammatical encoding does not actually alleviate stress and cause better performance. Indeed, as the effect is stronger for AWMU, this implies successfully realizing longer sentences relies on better strategy, as opposed to the alternative that they require more memory usage.

In other words, using less working memory is correlated with increased model fit, even after controlling for effects of sentence length or complexity. To reiterate, there are several possible explanations for this. For instance, pushing working memory to capacity could be more likely to cause errors, as speakers retrieve too many lexsyns that cannot be combined. In the model, this corresponds to flushing, where it throws away its current state upon realizing it cannot finish the sentence. The observable outcome in a language production system would be a disfluency: as discussed, human disfluency patterns are likely also caused by limitations in working memory. In turn, this model provides some hypotheses pointing to the types of strategies that may lead to certain kinds of disfluencies.

8.2. Increased Working Memory Capacity

Having additional working memory available improves model fit. This is an unsurprising result and is likely a simple consequence of certain sentences requiring more working memory to properly realize. Nonetheless, varying working memory capacity does not change the general strategy of grammatical encoding, which prefers to use less working memory and more right-branching constructions. Still, the model with less working memory was less right-branching: without additional working memory available, the model sometimes had to settle for an inferior strategy that attempted to build a sentence with non-incremental components too incrementally. We consider the lower fit of the lower working memory model to be commensurate with previous research, which leaves open as a possible avenue for future experimentation the correlation of lower working memory usage to higher fit.

Further, we do have evidence that the model is indeed making use of late decisions when it has more working memory available. This comes from the significant correlation of working memory with branching factor. A higher right-branching factor is associated with lower working memory use, as new elements are added to the current state, rather than built up in another way.

8.3. Incrementality in Constructions

We consider both of these results to be compatible with the hypothesis of strategic incrementality. More incremental processes require less working memory. This is because lexsyns can be combined and outputted, freeing space in working memory. Moreover, reducing working memory usage is normally used as a possible argument for why incremental strategies might be preferred. That still leaves two basic possibilities: (1) Speakers prefer to use constructions that are possible to realize more incrementally (i.e., the finding is about the type of sentence), or (2) speakers attempt to realize all constructions as incrementally as possible (i.e., the finding is about the speakers' strategy). We have reason to believe, from Ferreira and Swets (2002), that (2) is not the case, unless the speakers are under some stress to speak as quickly as possible. We leave (1) to future work.

8.4. Broader Implications

By limiting the model's working memory directly, we are able to demonstrate that working memory is critical to grammatical encoding: limiting it increases errors in behavioral studies and reduces the fit of our model. Our task was naturally not perfectly analogous to experimental work: experimenters do not have the option of directly limiting memory or measuring actual memory usage. However, our evidence on decreased model fit from limiting working memory meshes nicely with the existing experimental evidence on speech errors caused by limited working memory (Hartsuiker and Barkhuysen, 2006, Badecker and Kuminiak, 2007, Slevc, 2011).

Conversely, each sentence may dictate a minimum amount of working memory needed to realize a sentence without disfluencies, even with an incremental strategy. In that case, the model makes a prediction that can be tested by future experiments: sentences with lower minimum working memory requirements will have fewer production errors.

The empirical evaluation of our models suggest that there is a specific amount of modality-specific working memory available to speakers for grammatical encoding, and that speakers generally do not maximize working memory use. Importantly, this conclusion depends on the assumptions of ACT-R's memory account and CCG's general representational model of the syntactic process.

9. Limitations and Future Work

There are a few clear limitations to this approach that can possibly be addressed in future work. While the presented model succeeds in integrating an empirically validated theory of memory with an interpretable linguistic formalism and being evaluated on corpus data, the model's actual output is rarely globally idiomatic. This in of itself is a result: general declarative memory activations, production rule utilities, and grammar constraints do not seem to be sufficient to explain the entire grammatical encoding process. Its very likely that a more successful model will also need a component that more explicitly learns sequential information, such as with a Hidden Markov Model or Recurrent Neural Network.

Moreover, there are many other models that explain similar effects using a similar methodology but with different formalisms, such as PCFG surprisal or Gibson DLT (e.g., Rajkumar et al., 2016). This work is difficult to directly compare for various reasons, including the datasets used (written vs. spoken English), but also their lack of simulation or cognitive model. Nonetheless, in the future, a more thorough investigation has the potential to integrate the multitude of phonemona that have been studied in its relationship to syntactic choice.

10. Conclusion

In summary, we use a modeling & simulation approach to ask questions about the role of working memory in grammatical encoding. This starts by proposing an implemented computational-cognitive model of two of the stages of grammatical encoding: function assignment and constituent assembly. Our implementation combines a grammar formalism, Combinatory Categorial Grammar, and a cognitive architecture, ACT-R. We examined working memory's role during this stage of language production and how working memory limitations relate to incremental production. We evaluate our model on relatively large corpus data, which is novel in cognitive modeling. This evaluation revealed the model's fit increases with higher incrementality and lower working memory usage, but that having additional working memory available improves overall fit. Finally, our corpus-based evaluation of a cognitive model introduces a paradigm of inquiry that makes progress in modeling by comparing generative fits across different model versions.

Acknowledgements

This project was supported by the National Science Foundation (BCS-1457992). We thank Frank E. Ritter and Matthew Kelly for their comments on an earlier version.

References

Daniel J Acheson and Maryellen C MacDonald. 2009. Verbal working memory and language production: Common approaches to the serial ordering of verbal information. *Psychological bulletin* 135(1):50.

John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Quin. 2004. An integrated theory of the mind. *Psychological Review* 111:1036–1060.

Alan Baddeley. 1992. Working memory. *Science* 255(5044):556–559.

Alan Baddeley. 2003. Working memory: looking back and looking forward. *Nature Reviews Neuroscience* 4(10):829–839.

William Badecker and Frantisek Kuminiak. 2007. Morphology, agreement and working memory retrieval in sentence production: Evidence from gender and case in slovak. *Journal of Memory and Language* 56(1):65–85.

Jerry Ball, Andrea Heiberg, and Ronnie Silber. 2007. Toward a large-scale model of language comprehension in ACT-R 6. In *Proceedings of the 8th International Conference on Cognitive Modeling, Ann Arbor, Michigan.* pages 173–179.

J Kathryn Bock and Willem J M Levelt. 2002. *Language production: Grammatical encoding*, Routledge, volume 5, pages 405–452.

Franklin Chang, Gary S Dell, and J Kathryn Bock. 2006. Becoming syntactic. *Psychological Review* 113(2):234–272.

Andrew RA Conway, Michael J Kane, Michael F Bunting, D Zach Hambrick, Oliver Wilhelm, and Randall W Engle. 2005. Working Memory Span Tasks: A Methodological Review and User's Guide. *Psychonomic Bulletin & Review* 12(5):769–786.

Meredyth Daneman and Patricia A Carpenter. 1980. Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior* 19(4):450–466.

Gary S Dell, Franklin Chang, and Zenzi M Griffin. 1999. Connectionist models of language production: Lexical access and grammatical encoding. *Cognitive Science* 23(4):517–542.

Fernanda Ferreira and Karl GD Bailey. 2004. Disfluencies and human language comprehension. *Trends in cognitive sciences* 8(5):231–237.

Fernanda Ferreira and Benjamin Swets. 2002. How incremental is language production? Evidence from the production of utterances requiring the computation of arithmetic sums. *Journal of Memory and Language* 46(1):57–84.

Victor S. Ferreira. 1996. Is it better to give than to donate? Syntactic flexibility in language production. *Journal of Memory and Language* 35:724–755.

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

John J Godfrey, Edward C Holliman, and Jane McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-92)*. IEEE, volume 1, pages 517–520.

Markus Guhe. 2007. *Incremental Conceptualization for Language Production*. Lawrence Erlbaum Associates.

Robert J Hartsuiker and Pashiera N Barkhuysen. 2006. Language production and working memory: The case of subject-verb agreement. *Language and Cognitive Processes* 21(1-3):181–204.

Aravind K Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of formal languages*, Springer, pages 69–123.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*. volume 10, page 707.

Richard L. Lewis and Shravan Vasishth. 2005. An activation-based model of sentence processing as skilled memory retrieval. *Cognitive science* 29(3):375–419.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop in Barcelona*. pages 74–81.

Martin J. Pickering and Holly P. Branigan. 1998. The representation of verbs: Evidence from syntactic priming in language production. *Journal of Memory and Language* 39:633–651.

Rajakrishnan Rajkumar, Marten van Schijndel, Michael White, and William Schuler. 2016. Investigating locality effects and surprisal in written english syntactic choice phenomena. *Cognition* 155:204–232.

David Reitter, Frank Keller, and Johanna D. Moore. 2011. A Computational Cognitive Model of Syntactic Priming. *Cognitive Science* 35(4):587–637.

L Robert Slevc. 2011. Saying what's on your mind: Working memory effects on sentence production. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 37(6):1503.

Mark Steedman. 2000a. Information structure and the syntax-phonology interface. *Linguistic Inquiry* 31(4):649–689.

Mark Steedman. 2000b. *The Syntactic Process*. MIT Press, Cambridge, MA.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell* .

Niels A Taatgen, Christian Lebiere, and John R Anderson. 2006. Modeling paradigms in act-r. *Cognition and multi-agent interaction: From cognitive modeling to social simulation* pages 29–52.

Hedderik Van Rijn and John R Anderson. 2003. Modeling Lexical Decision as Ordinary Retrieval. In *Proceedings of the fifth international conference on cognitive modeling*. pages 207–212.

Margreet Vogelzang, Anne C. Mills, David Reitter, Jacolien Van Rij, Petra Hendriks, and Hedderik Van Rijn. 2017. Toward cognitively constrained models of language processing: A review. *Frontiers in Communication* 2(11).