Left-corner parsing of Minimalist Grammars

Tim Hunter

April 3, 2018

Abstract

Much recent research in experimental psycholinguistics revolves around the resolution of long-distance dependencies, and the manner in which the human sentence processor "retrieves" elements from earlier in a sentence that must be related in some way to the material currently being processed. At present there is no obvious way for the issues raised by this research to be framed in terms of an MG parser. Stabler's 2013 top-down MG parser does not involve any corresponding notion of "retrieval": it requires that a phrase's position in the derivation tree be completely identified before the phrase can be scanned, which means that a filler cannot be scanned without committing to a particular location for its corresponding gap. In this paper I will attempt to develop a parsing algorithm that is inspired by Stabler 2013, but which allows a sentence-initial filler to be scanned immediately while delaying the choice of corresponding gap position.

Much recent research in the experimental psycholinguistics literature revolves around the resolution of long-distance dependencies, and the manner in which the human sentence processor "retrieves" elements from earlier in a sentence that must be related in some way to the material currently being processed. A canonical instance is the resolution of a wh-dependency, where a filler wh-phrase must be linked with an associated gap site; in this case it is now well-established, for example, that humans actively predict gap sites in advance of definitive confirming bottom-up evidence. At present, however, there is no obvious way for discussion of these findings to be framed in terms of an MG parser. Stabler (2013) presents a top-down MG parser that is incremental, but does not involve any corresponding notion of "retrieval": it requires that a phrase's position in the derivation tree be completely identified before the phrase can be scanned, which has the consequence that a filler cannot be scanned without committing to a particular location for its corresponding gap.

In this paper I will attempt to develop a parsing algorithm that is inspired by Stabler 2013, but which allows a sentence-initial filler (such as a wh-word) to be scanned immediately while delaying the choice of corresponding gap position. In addition to mixing bottom-up with top-down information in the familiar manner of left-corner parsers, the crucial innovation is in allowing the various features of a moving phrase to be checked at different points in the parser's progress: for example, allowing a filler's wh-feature to be checked without checking its theta-role-feature (or equivalent) at the same time. This requires enriching the usual stack to allow categories that have only some of their features checked to "wait in the background", while normal processing in the manner of a context-free left-corner parser moves from the filler position, where the waiting phrase had its movement features checked, to the gap position, where the waiting phrase has its base-position features checked.

The resulting system, as it stands, naturally handles constructions involving movement of constituents that do not themselves contain traces. Movement of constituents containing traces (i.e. remnant movement) creates complications that I have not completely resolved, but I outline one approach to these puzzles that handles some cases and seems to have the potential to generalize.

Section 1 very briefly reviews some relevant experimental findings involving retrieval, as motivation.

I describe the top-down MG parser from Stabler (2013) in Section 2, point out the difficulties in connecting this to questions about retrieval, and then present an initial attempt at a left-corner MG parser that avoids these difficulties in Section 3. Some extensions are required to deal with remnant movement, which I motivate and introduce in Section 4; these extensions accommodate at least some remnant movement configurations, but their consequences are not yet fully understood. While still incomplete, the system presented here does already point towards some interesting new questions about the subtleties of the experimental findings, which I discuss in Section 5.

1 Human processing of long-distance dependencies

1.1 Empirical background

A significant problem that confronts the human sentence-processor is the treatment of *filler-gap* dependencies. These are dependencies between a pronounced element, the *filler*, and a position in the sentence that is not indicated in any direct way by the pronunciation, the *gap*. A canonical example is the kind of dependency created by wh-movement, for example the one shown in (1).

(1) What did John buy ____ yesterday?

The interesting puzzle posed by such dependencies is that a parser, of course, does not get to "see" the gap: it must somehow determine that there is a gap in the position indicated in (1) on the basis of the properties of the surrounding words, for example the fact that 'what' must be associated with a corresponding gap, the fact that 'buy' takes a direct object, etc.

Experimental psycholinguistic work has uncovered a number of robust generalizations about how the human parsing system decides where to posit gap sites in amongst the pronounced elements as it works through a sentence incrementally. In most standard cases humans follow the "active gap-filling" strategy (Fodor, 1978; Stowe, 1986): that is, hypothesize that there is a gap in any position where there might be one, and retract this hypothesis if the next word provides bottom-up evidence disconfirming it. (This is intuitively perhaps the "safe" strategy, erring on the side of the easily disconfirmed hypothesis to be sure that we leave no stone unturned.) Specifically, evidence suggests that the dependency in (1) is constructed before the parser encounters 'yesterday'. This conclusion is based on the so-called "filled-gap effect": in a sentence like (2), we observe a reading slowdown at 'a book' (Stowe, 1986).

(2) What did John buy a book about ____ yesterday?

This slowdown is what one might expect if a dependency between 'what' and the object-position of 'buy' were constructed before the subject reads past 'buy', and then had to be retracted when 'a book' is read. (What was hypothesized to be a gap position is in fact filled, hence "filled-gap effect".)

This basic generalization prompts a number of questions about the details of when and how this sort of hypothesizing of a gap takes place, and in particular the way this mechanism interacts with the intricate grammatical constraints upon the relevant long-distance dependencies; I mention just a few instances to illustrate. For example, it has been shown that active gap-filling is island-sensitive: readers do not pre-emptively posit a gap after 'wrote' in (3), in keeping with the fact that complete sentences with an analogous gap in that position, such as (4), are unacceptable (Traxler and Pickering, 1996).

(3) What did [the author who wrote the book]$_{\text{island}}$ see ____ yesterday?

(4) * What did [the author who wrote ____]$_{\text{island}}$ see a movie yesterday?

Interestingly, the active gap-filling strategy *does* extend to parasitic gaps that precede their licensing gap: readers pre-emptively create a link between 'what' and the object position of 'repair' in examples like (5), despite the fact that this would be ill-formed without the accompanying gap after 'destroy' (Phillips, 2006). This strategy does *not*, however extend to parasitic gaps that follow their licensing gap: there is no pre-emptive dependency creation at 'researching' in (6) (Wagers and Phillips, 2009).

(5) What did the attempt to repair ____ ultimately destroy ____?

(6) What did John attempt to repair ____ before researching ____?

More recent research has investigated analogous questions about different kinds of long-distance dependencies besides filler-gap dependencies, such as the dependency between a pronoun or reflexive and its antecedent (e.g. Dillon, 2014), and the dependency between an NPI and its licensor (e.g. Parker and Phillips, 2016). These other kinds of dependencies bring with them their own grammatical constraints on the structural locations that they can link, raising questions along the lines of the island-related questions for wh-movement. They also differ in the degree to which pronounced elements at each "end" of the dependency provide evidence that the dependency must be posited: in the classical filler-gap dependency, the filler *must* be connected to something further downstream, but the location of the something must be guessed; a pronoun, by contrast, may or may not require linking to an antecedent elsewhere in the sentence, either earlier or later; a reflexive (or NPI) does require such a link to an antecedent (or negative element) that generally comes earlier in the sentence, but unlike a wh-filler this earlier element doesn't trigger any such requirement.

The term "retrieval" has come to be used as a relatively general-purpose name for the mechanisms that create these kinds of long-distance dependencies[1], at the latter of the two positions being related, linking the current position in the sentence to some element from earlier that is said to be retrieved. The idea that there is something in common across all these different kinds of dependencies comes from the fact that their grammatical descriptions seem to have c-command as a common ingredient.

1.2 Retrieval decisions as ambiguity-resolution

At present it is difficult for the generalizations emerging from the experimental work just discussed to be framed in terms of the workings of a comprehensive MG parser. Consider for comparison the earlier empirical work on attachment preferences and garden path theory (e.g. Frazier and Clifton, 1996): since the focus was on grammatical relationships that were local in phrase-structural terms (e.g. the dependency between an adjunct PP and its host, or between a verb and its optional object), the strategies being discovered could be understood as strategies for searching through the hypothesis space induced by the operations of a context-free parser. For example, the garden-path effect in (7) can be interpreted as evidence that given the locally ambiguous prefix 'When the dog scratched the vet', readers pursue the analysis in (8a) rather than the one in (8b). This is an instance of the Late Closure preference.

[1]The descriptor "long-distance" is generally understood to mean dependencies that are non-local relative to a surface structure tree. For example, an agreement dependency between the head of a large NP subject and the verb that follows it would not typically be thought of as long-distance in this sense, even if there is a large linear distance separating the head noun from the verb (in terms of number of words), because it is a realization of the local relationship between two sisters in a headed/endocentric phrase structure tree. One of the questions that the work described in this paper aims to bring into sharper focus is to what degree these two kinds of "linearly long-distance" dependencies need to be thought of separately.

(7) When the dog scratched the vet and his new assistant removed the muzzle.

(8) a. When [$_S$ the dog scratched the vet] [$_S$ ...]
b. When [$_S$ the dog scratched] [$_S$ the vet ...]

Another way to put this is to say that after the word 'scratched', a bottom-up parser has the choice between performing a reduce step (to analyze this verb as a complete, intransitive VP) or performing a shift step (supposing that other remaining input will also be part of the VP), and it prefers the latter.

In principle, it should be possible to give an analogous description of the active gap-filling generalization: we can imagine a description of the parser's search space that allows us to state preferences for one kind of transition (the kind that interrupts "local processing" and performs retrieval of a filler, roughly) over another (the kind that continues working with local material, roughly). This is difficult at present, however, because there are relatively few formal models of parsing that treat both long-distance dependencies and local dependencies in a cohesive, integrated manner. Aside from this technical hurdle, however, the active gap-filling generalization can be seen as one with the same form as the Late Closure preference: just as humans' first guess given the prefix 'When the dog scratched the vet' is (8a) rather than (8b), their first guess given the prefix 'What did John buy' is (9a) rather than (9b).

(9) a. What did John buy ____ ...
b. What did John buy ...

One incremental parser that does treat both long-distance dependencies and local dependencies is the top-down MG parser of (Stabler, 2013). It is difficult to connect the empirical generalizations about when retrieval occurs to the choices faced by this parser, however, because it doesn't involve any appropriate notion of retrieval: as I will discuss in Section 2, before consuming a filler (e.g. a wh-phrase) it must commit to a particular location for its corresponding gap to appear.[2] This means that there is no choice point of the sort mentioned above, where the parser must decide between performing retrieval to create some long-distance dependency involving the current position and continuing on without doing so.

2 The top-down MG parser

I will assume familiarity with the MG formalism (Figure 1) (Stabler, 1997; Stabler and Keenan, 2003), and in particular with the close relationship between MGs and MCFGs (Michaelis, 2001). I will use MCFG-style derivation trees, of the sort shown in Figure 2, throughout. Each node is labeled with a tuple of feature-sequences, and the size of this tuple corresponds to the arity of the MCFG nonterminal; I will call such a tuple a category, and if the length of the tuple is one I will call it a singleton category. Each leaf node must be labeled by a singleton category, and must be accompanied by a string; the lexicon is a list of allowed pairings of a singleton category with a string. The category labeling each unary-branching node must be derivable from the category labeling the single daughter node in accord with the MG operation MOVE. The category labeling each binary-branching node must be derived from the categories labeling the two daughter nodes in accord with the MG operation MERGE. (And there are no nodes with more than two daughters.)

Stabler's (2013) top-down MG parser adapts the idea underlying the straightforward top-down CFG parser to this MCFG-like understanding of MGs. We begin with the start category — taking `C` to be

[2] In principle, multiple hypotheses might be pursued in parallel, of course, but each one will be committed to a particular gap location.

Given an alphabet Σ and a set of basic types Ty, the set of features is $Feat = \bigcup\{\{x, \texttt{=}x, \texttt{+}x, \texttt{-}x\} | x \in Ty\}$, and the set of expressions is $Expr = (\Sigma^* \times Feat^*) \times (\Sigma^* \times Feat^*)^*$. Given a finite lexicon $Lex \subset \Sigma^* \times Feat^*$, merge is a binary partial function on $Expr$ and move is a partial unary function on $Expr$, defined as follows where $s, t \in \Sigma^*$, $\alpha, \beta \in Feat^*$, $\gamma \in Feat^+$, $\texttt{f} \in Ty$, $\phi, \psi \in (\Sigma^* \times Feat^*)^*$.

MERGE = COMP-MERGE $\cup$ SPEC-MERGE $\cup$ NONFINAL-MERGE

$$\frac{s :: \texttt{=f}\alpha \in Lex \qquad t :: \texttt{f}, \psi}{st :: \alpha, \psi}\ \text{COMP-MERGE} \qquad \frac{s :: \texttt{=f}\alpha, \phi \notin Lex \qquad t :: \texttt{f}, \psi}{ts :: \alpha, \phi\psi}\ \text{SPEC-MERGE}$$

$$\frac{s :: \texttt{=f}\alpha, \phi \qquad t :: \texttt{f}\gamma, \psi}{s :: \alpha, \phi(t, \gamma)\psi}\ \text{NONFINAL-MERGE}$$

MOVE = SPEC-MOVE $\cup$ NONFINAL-MOVE

$$\frac{s :: \texttt{+f}\alpha, \phi(t, \texttt{-f})\psi}{ts :: \alpha, \phi\psi}\ \text{SPEC-MOVE} \qquad \frac{s :: \texttt{+f}\alpha, \phi(t, \texttt{-f}\gamma)\psi}{s :: \alpha, \phi(t, \gamma)\psi}\ \text{NONFINAL-MOVE}$$

where we require, in both cases, that there is no other $(t', \texttt{-f})$ or $(t', \texttt{-f}\gamma')$ in ϕ or ψ

Figure 1

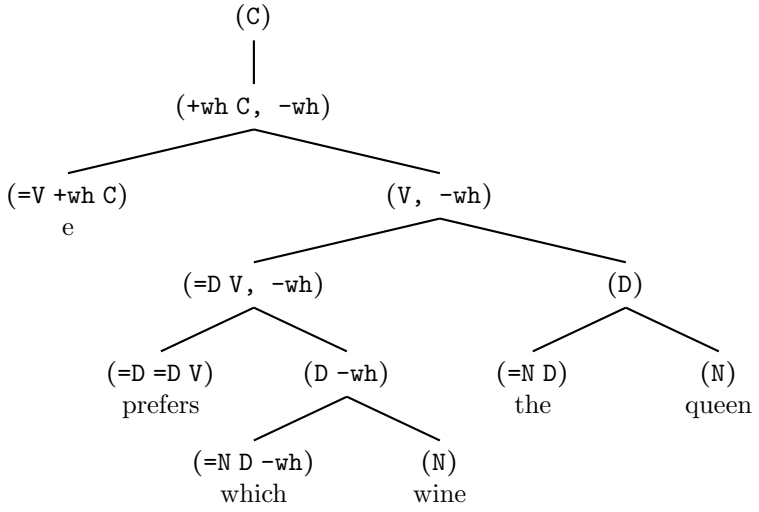


Figure 2

$_0$ which $_1$ wine $_2$ the $_3$ queen $_4$ prefers $_5$

Step	Rule	Counter	Queue
0		0	$(\texttt{C})$
1	spec-mv	0	$(\texttt{+wh C}, \texttt{-wh}_0)_1$
2	comp-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{V}, \texttt{-wh}_0)_{11}$
3	spec-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D V}, \texttt{-wh}_0)_{111}$, $(\texttt{D})_{110}$
4	nonfinal-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D =D V})_{111}$, $(\texttt{D -wh})_0$, $(\texttt{D})_{110}$
5	comp-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D =D V})_{111}$, $(\texttt{=N D -wh})_{00}$, $(\texttt{N})_{01}$, $(\texttt{D})_{110}$
6	scan	1	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D =D V})_{111}$, $(\texttt{N})_{01}$, $(\texttt{D})_{110}$
7	scan	2	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D =D V})_{111}$, $(\texttt{D})_{110}$
8	scan	2	$(\texttt{=D =D V})_{111}$, $(\texttt{D})_{110}$
9	comp-merge	2	$(\texttt{=D =D V})_{111}$, $(\texttt{=N D})_{1100}$, $(\texttt{N})_{1101}$
10	scan	3	$(\texttt{=D =D V})_{111}$, $(\texttt{N})_{1101}$
11	scan	4	$(\texttt{=D =D V})_{111}$
12	scan	5	ϵ

Table 1

the distinguished starting feature of the MG, this will be the singleton category (C) — and rewrite in a top-down manner, expanding the frontier of the tree by replacing one category at a time. Each such predictive rewriting step corresponds exactly to either one of the three subcases of MERGE (if it introduces two new categories) or to one of the two subcases of MOVE (if it introduces one new category). The first few steps shown in Table 1 (ignoring the subscripts for now) are examples of this, carrying out the first few top-down expansions starting from the root of the tree in Figure 2. When such predictive rewriting steps eventually introduce a category that the lexicon pairs with the first word waiting in the input buffer, that word will be scanned, and the category will be removed from the parser's record of the frontier of the tree — that prediction has been fulfilled. What remains is a record of that part of the frontier of the tree that is yet to be matched with input.

In the context-free case, a top-down parser can perform what is in effect a leftmost derivation, always expanding the leftmost category in its store, since this is guaranteed to be the one category on the frontier that dominates the next word to be consumed. (The parser's store therefore operates as a stack, with the "top" of the stack corresponding to the "leftmost" symbol on the frontier.) What makes things more complicated in the MG top-down parser is that this simple relationship no longer holds between the order of the category symbols that make up the current frontier of the tree and the order of the incoming words: in the tree in Figure 2, the path from the root node down to the terminal node 'which' is *not* simply the path obtained by stepping down to the leftmost daughter at each point. This is a direct consequence of the way MCFGs generalize beyond the simple situation that holds in a CFG, where the yield of a tree with two daughter subtrees is the yield of the left daughter concatenated with the yield of the right daughter.

Turning to the specifics of Table 1: notice that of the two categories in the store in Step 2 — corresponding to the frontier of the tree as it stands after two expansions — it is the *rightmost* one, that is expanded next, according to the rule

$$(\texttt{V, -wh}) \quad \rightarrow \quad (\texttt{=D V, -wh}) \quad (\texttt{D})$$

to reach the configuration shown in Step 3. Of the two categories thereby introduced, it is the leftmost one (the middle one of the three shown in Step 3) that is expanded next, according to the rule

$$(\texttt{=D V, -wh}) \quad \rightarrow \quad (\texttt{=D =D V}) \quad (\texttt{D -wh})$$

to reach Step 4. Next the rightmost of these is expanded, to result in a situation where the tree has been expanded down to the frontier

$$\texttt{(=V +wh C)} \quad \texttt{(=D =D V)} \quad \texttt{(=N D -wh)} \quad \texttt{(N)} \quad \texttt{(D)}$$

in Step 5, and the next step can now scan the first word of the input, 'which', to fulfill the prediction represented by the third category in this frontier, which is then removed in Step 6.

Since the tree-based order in which the categories are shown in Table 1 does not suffice to answer the question of "where to look next" in the way it does for a simple context-free parser, the MG parser must track some more information about the relationships amongst the various derivational subconstituents that are in its store. Specifically, it must track the relative ordering of the *derived positions* of those various subconstituents. This extra information is in the subscripts that we have ignored until now. The subscripts are Gorn addresses into the derived tree. Notice that in Step 5, `(=N D -wh)` has the leftmost Gorn address of the five categories present (namely 00); it is for this reason that it (and not any other category in the store) is a candidate for matching a word from the input buffer. Having the leftmost Gorn address corresponds to being on the top of the stack in the context-free case: that is where the parser looks to work next, whether that work is scanning a word from the input buffer or rewriting a category according to some other rule. To take an example of the latter, the fact that `(D -wh)` has the leftmost Gorn address (namely 0) of the four categories present in Step 4 is what dictates that this, rather than any other category, should be expanded at the next step. Notice that when `(D -wh)` is expanded, the address of the new left daughter is obtained by appending 0 to the parent address (producing 00), and the address of the new right daughter is obtained by appending 1 (producing 01).

In general, however, the notion of the "derived position" of a particular derivational subconstituent is not so straightforward. It is relatively simple in the cases just discussed (Steps 4 and 5), because all the categories involved there are singleton categories: they have no moving subparts, and therefore correspond directly to subtrees of the derived tree (even if their order in the derivation tree differs from the order of the corresponding subtrees of the derived tree). With complex, multi-component categories the situation is different: notice that the two-component categories in Steps 2–4 have an additional address for their moving `-wh` components. For example, in Step 2 we have $(\texttt{V}, \texttt{-wh}_0)_{11}$, indicating a completed `V` projection out of which a `-wh` phrase will move, where the `V` projection's address in the derived tree is 11 and the `-wh` phrase's address is 0. This address was given to the `-wh` component when it was first introduced in Step 1, as part of the rewriting of the start symbol `(C)`: in derivation-tree terms this is a unary rewriting step that replaces `(C)` with the category `(+wh C, -wh)`, but the consequence of this in derived-tree terms is that the completed `C` projection that we have predicted will appear at address ϵ (i.e. the root node) will have a phrase that has moved to check a `-wh` feature as its left daughter (address 0), and will have all the other, non-moving parts of the `(+wh C, -wh)` expression somewhere within its right daughter (address 1). The moving subcomponent with address 0 stands by as the non-moving parts are expanded in Steps 2 and 3, and then in Step 4 the 0 becomes the full-fledged address of a singleton category: this is the crucial step where we decide to expand `(=D V, -wh)` as an instance of the nonfinal-MERGE rule, meaning that one of the daughters of this node *is* the moving `-wh` subcomponent.

The crucial point to notice for present purposes is that a word can only be scanned once the full root-to-leaf path to that word in the derivation tree has been expanded. Committing to such a sequence of expansions commits the parser to particular choices about where *each* of that word's features are checked. In Table 1, expansions that precede the first scan step involve the checking of a `-wh` movement feature at (in derived tree terms) the specifier of the root CP *and* the checking of a `D` selectional feature at the object position of the matrix clause. Therefore when 'which' is scanned at Step 6, it is being scanned *as* (the head of) a wh-phrase that moves from the matrix object position to the matrix SpecCP position; the steps taken down to and including Step 6 are not compatible with any other base (pre-movement) position for the phrase headed by 'which'.

$_0$ which $_1$ wine $_2$ prefers $_3$ the $_4$ queen $_5$

Rule	Counter	Queue
	0	(C)
spec-mv	0	$(\texttt{+wh C}, \texttt{-wh}_0)_1$
comp-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{V}, \texttt{-wh}_0)_{11}$
nonfinal-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D V})_{110}$, $(\texttt{D -wh})_0$
comp-merge	0	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D V})_{110}$, $(\texttt{=N D -wh})_{00}$, $(\texttt{N})_{01}$
scan	1	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D V})_{110}$, $(\texttt{N})_{01}$
scan	2	$(\texttt{=V +wh C})_{10}$, $(\texttt{=D V})_{110}$
⋮	⋮	⋮

Table 2

To illustrate the point, consider Table 2 where we have a fronted wh-phrase that has moved from the matrix subject position instead. Only an initial portion of the parser's behaviour is shown, but we can see that the expansion steps that lead up to the scanning of 'which' (and also the scanning of 'book' at the very next step) are different from those that lead up to this first scan step in Table 1. Only the first two steps are shared, which correspond to breaking down the sentence into the wh-attracting complementizer (category (=V +wh C)) and a V projection out of which a wh-phrase will move (category (V, -wh)). Beyond this, the parser must choose between expanding the latter in a way that takes the moving wh-phrase the subject (as in Table 2) and doing so in a way that takes the moving wh-phrase to be somewhere further embedded. So the parser must make a choice between these two analyses even before scanning the word 'which'.

The underlying assumption running through discussions of the experimental results reviewed above, however, is that it is possible to scan a filler (here, 'which wine') while remaining uncomitted to any particular corresponding gap site. Having scanned a filler at the left edge of a clause, interesting questions arise about how a parser can subsequently — as it reads through the rest of the sentence — decide where to posit the corresponding gap, and these questions are the target of investigation in those experimental studies. This top-down MG parser, however, does not make available any corresponding notion of deciding where to discharge a pending obligation to posit a gap site, because the decision has already had to be made before the filler which creates this obligation could be scanned. The identification of filler sites is inseparable from the identification of gap sites. This is the issue that motivates the search for a left-corner based parsing strategy described in the next section.

3 Towards a left-corner MG parser

In order to improve upon the top-down MG parser in the sense just discussed, I take inspiration from the left-corner parsing strategy for context-free grammars. I provide a quick review of context-free left-corner parsing, largely to fix some notation and terminology, before turning to the left-corner MG parser that I propose.

3.1 CFG left-corner parsing

Assume a context-free grammar with nonterminal alphabet N and terminal alphabet V, with start symbol $\mathrm{S} \in N$. Then the left-corner parsing schema defines a transition relation on *configurations*, where a configuration is an ordered pair: the first component of a configuration, the *stack*, is a

sequence of elements of $\{X : X \in N\} \cup \{\overline{X} : X \in N\}$, and the second component, the *buffer*, is a sequence of elements of V. For a given $w_1 w_2 \dots w_n \in V^*$ to be parsed, the starting configuration is $(\overline{\mathrm{S}}, w_1 w_2 \dots w_n)$, and the goal configuration is (ϵ, ϵ). The transition relation is the union of the four relations defined in (10).[3] (I make the simplifying assumption that each production rule has on its right hand side either a single terminal symbol or a sequence of nonterminal symbols, not a mixture of the two.)

(10) For $w_i \in V$, $X, Y_i \in N$, $\alpha \in (\{X : X \in N\} \cup \{\overline{X} : X \in N\})^*$

a. $(\alpha, w_i \dots w_n) \xrightarrow{\text{shift}} (X\alpha, w_{i+1} \dots w_n)$
where $X \to w_i$ is a rule of the grammar

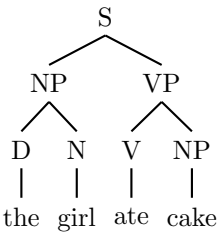
b. $(X\alpha, w_i \dots w_n) \xrightarrow{\text{scan}} (\alpha, w_{i+1} \dots w_n)$
where $X \to w_i$ is a rule of the grammar

c. $(Y_1\alpha, w_i \dots w_n) \xrightarrow{\text{predict}} (\overline{Y_2} \dots \overline{Y_m} X\alpha, w_i \dots w_n)$
where $X \to Y_1 \dots Y_m$ is a rule of the grammar

d. $(Y_1\overline{X}\alpha, w_i \dots w_n) \xrightarrow{\text{connect}} (\overline{Y_2} \dots \overline{Y_m}\alpha, w_i \dots w_n)$
where $X \to Y_1 \dots Y_m$ is a rule of the grammar

Table 3 shows how these rules operate on the sentence whose derivation is shown in (11). (The grammar is left implicit.)

(11)



Nonterminal symbols with a bar over them correspond to unfulfilled predictions; hence the start configuration containing the symbol $\overline{\mathrm{S}}$. The shift rule consumes a word from the input, and adds a symbol to the stack indicating the appropriate bottom-up, already-recognized nonterminal (i.e. the symbol has no bar over it). The scan rule also consumes a word from the input, but uses it to remove a symbol from the stack, specifically a predicted instance of a nonterminal that the word can satisfy. The predict rule and the connect rule both "trade in" a bottom-up recognized nonterminal (no bar) for some number of predictions (with bars) corresponding to that nonterminal's hypothesized sisters, according to some chosen rule. When predict applies in Step 2 of Table 3, for example, the already-recognized D is hypothesized to be the left-corner of an NP constituent expanded according to the rule 'NP → D N'; it is therefore traded in for a predicted N, accompanied by a bottom-up NP symbol that will be available to work with if that prediction of an N is fulfilled. When connect applies in Step 4, the recognized NP (left corner of the rule 'S → NP VP') is similarly traded in for a predicted VP; what makes the connect rule different is that we put this recognized NP towards the satisfaction of an *already predicted* instance of the parent nonterminal (here, $\overline{\mathrm{S}}$), so we remove this symbol from the stack rather than adding a bottom-up instance of this parent nonterminal.

The guiding intuition behind the MG parser proposed below is that it should consume a filler (e.g. a wh-phrase), and update its predictions accordingly in something like the way a context-free left-corner parser would when working with a GPSG-style rule like S → WH $\mathrm{S}_{[/\mathrm{wh}]}$. An illustrative example is shown in (12).

[3] This presentation is closely based on that of Kanazawa (2016).

Step	Rule		Remaining input	Stack
0			the girl ate cake	$\overline{\text{S}}$
1	shift	D → the	girl ate cake	D $\overline{\text{S}}$
2	predict	NP → D N	girl ate cake	$\overline{\text{N}}$ NP $\overline{\text{S}}$
3	scan	N → girl	ate cake	NP $\overline{\text{S}}$
4	connect	S → NP VP	ate cake	$\overline{\text{VP}}$
5	shift	V → ate	cake	V $\overline{\text{VP}}$
6	connect	VP → V NP	cake	$\overline{\text{NP}}$
7	scan	NP → cake	ϵ	ϵ

Table 3

(12)

Step	Rule		Remaining input	Stack
0			what did the girl eat	$\overline{\text{S}}$
1	shift	WH → what	did the girl eat	WH $\overline{\text{S}}$
2	connect	S → WH $\text{S}_{[/\text{wh}]}$	did the girl eat	$\overline{\text{S}_{[/\text{wh}]}}$

The crucial point here is that the parser consumes the filler, and updates its state accordingly to reflect the fact that it is now expecting a "gapped sentence" to follow, without yet committing to any particular structural position for the gap. Recall that this contrasts with the way the top-down MG parser discussed above must decide on *all* the positions in which a filler phrase checks features before that phrase can be scanned.

3.2 Basic cases

We can carry over to the MG setting the distinction between bottom-up recognized elements (e.g. the wh-phrase in (12), represented by symbols without bars) and top-down predicted elements. In addition, we will need to separate out the different feature-checking steps that a particular constituent is involved in in the course of a derivation: for example, given a phrase of category `(D -wh)`, we wish to be able to talk about the `D`-position of the `-wh`-position of this phrase, in either a bottom-up or top-down manner. This second step crucially differs from the workings on the top-down MG parser.

To begin, Table 4 shows a left-corner parse of the simple derivation shown in Figure 3. Since there is no movement yet, we do not yet need to make use of the separation of the feature-checking steps for moving elements. For clarity of exposition I take lexical items that are usually phonologically null to have overt pronunciations (usually 'e'). In addition to shift and scan rules that are analogous to the rules of the same name in the CFG case, the left-corner MG parser has a connect transition rule and a predict transition rule for each of the five MG rules (the three cases of merge and two cases of move as shown in Figure 1). The full array of these ten transition rules is shown below in Figure 6. For now, the only two MG rules relevant to the simple movement-free derivation in Figure 3 are comp-mrg and spec-mrg; the parse in Table 4 makes use of the comp-merge-connect, comp-merge-predict and spec-merge-connect transition rules.

The first two steps in Table 4 mimic the way a CFG left-corner parser would work with the rules

`(C)` → `(=V C)` `(V)` (comp-merge)

`(=V C)` → e

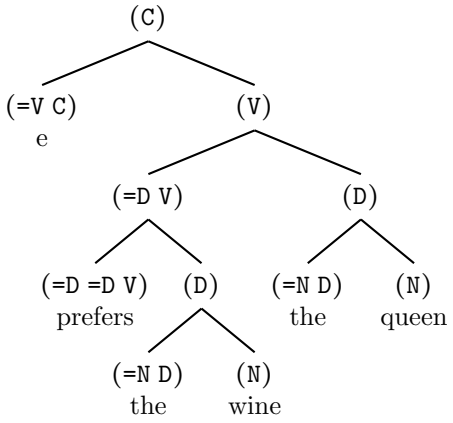


Figure 3: A derivation with no movement yet. The parse is shown in Table 4.

Step	**Rule**	**Input**	
0		e the queen prefers the wine	$\overline{\texttt{(C}\rceil}$
1	shift	the queen prefers the wine	$\texttt{(=V C}\rceil_1$ $\overline{\texttt{(C}\rceil}$
2	comp-mrg-connect	the queen prefers the wine	$\overline{\texttt{(V}\rceil}$
3	shift	queen prefers the wine	$\texttt{(=N D}\rceil_1$ $\overline{\texttt{(V}\rceil}$
4	comp-mrg-predict	queen prefers the wine	$\overline{\texttt{(N}\rceil}$ $\texttt{(D}\rceil_0$ $\overline{\texttt{(V}\rceil}$
5	scan	prefers the wine	$\texttt{(D}\rceil_0$ $\overline{\texttt{(V}\rceil}$
6	spec-mrg-connect	prefers the wine	$\overline{\texttt{(=D V}\rceil_0}$
7	shift	the wine	$\texttt{(=D =D V}\rceil_1$ $\overline{\texttt{(=D V}\rceil_0}$
8	comp-mrg-connect	the wine	$\overline{\texttt{(D}\rceil}$
9	shift	wine	$\texttt{(=N D}\rceil_1$ $\overline{\texttt{(D}\rceil}$
10	comp-mrg-connect	wine	$\overline{\texttt{(N}\rceil}$
11	scan	ϵ	ϵ

Table 4: Parsing the derivation with no movement shown in Figure 3.

namely shifting on the bottom-up recognized (=V C), and then connecting it to the predicted start category (C) to create a predicted (V). The 1 subscript on the shifted category indicates that the corresponding constituent is known to be lexical: this is tracked in case it becomes relevant to distinguishing between whether comp-mrg or spec-mrg can apply. A 0 subscript indicates that a constituent is known to be non-lexical; the absence of a subcript leaves the lexicality undetermined. (The unusual "hook" marks that appear inside the closing parenthesis of each category can be ignored for now; these are relevant to the separation between the checking of selectee features and licensee features, which will be illustrated below.)

Steps 3 and 4 in Table 4 mimic the way a CFG left-corner parser would work with the rules

$$\begin{array}{rcll} \texttt{(D)} & \rightarrow & \texttt{(=N D)} \quad \texttt{(N)} & \text{(comp-merge)} \\ \texttt{(=N D)} & \rightarrow & \text{the} & \end{array}$$

again shifting on the lexical category (=N D), but this time positing a new parent category (D) to follow the predicted sister (N), rather than connecting to an existing prediction as above. After the next input word 'queen' is scanned (Step 5) to fulfill this prediction, the now-established bottom-up (D) is connected (Step 6) to the subsequent top-down prediction according to the rule

$$\texttt{(V)} \quad \rightarrow \quad \texttt{(=D V)} \quad \texttt{(D)} \qquad \text{(spec-merge)}$$

to leave a predicted (=D V). Notice that although (D) appears as the right daughter in the tree (and in the rule above), because I am adopting the convention that the expression whose selector feature is being checked is always the left daughter of a merge step, we are taking it to be the linearly left daughter of the (V) (i.e. a specifier); therefore the resulting prediction has a 0 subscript marking it as non-lexical, with the effect that, unlike other top-down predictions, it *cannot* be satisfied with a scan step.

The remaining steps are similar to what has come before them.

We can now turn to a case involving one simple movement step. For comparison, I will use the derivation for which a top-down parse was discussed above. The derivation tree is repeated in Figure 4, this time with an additional dashed line: from now on I will include lines like this indicating the implicit dependency between the movement step at the top of the tree and the subtree rooted at the node labeled (D -wh). I will say that the (D -wh) node is a *quasi-daughter* of the root (C) node, and is a *quasi-sister* of the (+wh C,-wh) node. Notice that these relationships mirror the fact that the wh-phrase 'which wine' is a daughter of the root CP node, and sister to a C′ node, in the relevant derived tree. Making these relationships more salient is one step towards implementing the idea gestured at in (12).

A trace of the MG left-corner parser's construction of this derivation is shown in Table 5. The first three steps involve transitions we have seen before: specifically, they correspond to shifting a determiner, analyzing it as the left-corner of a DP, and then scanning the predicted N, just as in Steps 3–5 in Table 4. The additional -wh feature is irrelevant so far. As the "stack" in Step 3 indicates, we end up with a bottom-up recognized (D -wh) which we hope to put towards fulfilling a predicted (C).

Step 4 is a connect step: we connect this bottom-up constituent to the predicted (C). The connection being made is not one between a node and its mother in the tree in Figure 4, but rather — since this transition is based on an application of the spec-move rule — one between the (D -wh) node and its *quasi-mother*. Just as the connect step in Step 2 of Table 4 can be thought of as drawing the two lines that branch down from the (C) node in Figure 3, this instance of spec-mv-connect can be thought of as drawing the dashed line in Figure 4 and also the single solid line from the root (C) node down to its daughter. It is our version of the connect step in (12). It amounts to "deciding

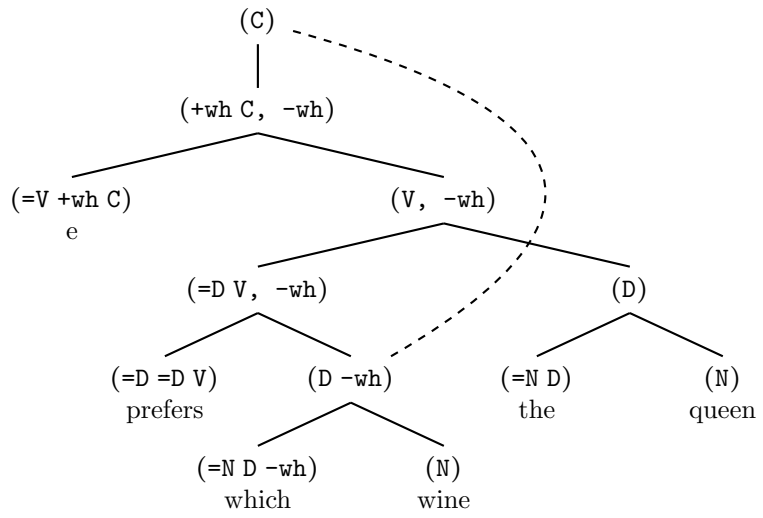


Figure 4: A derivation with one movement, from object position. The parse is shown in Table 5.

$_0$ which $_1$ wine $_2$ e $_3$ the $_4$ queen $_5$ prefers $_6$

Step	**Rule**	**Counter**	
0		0	$\overline{\texttt{(C⌋)}}$
1	shift	1	$\texttt{(=N D -wh⌋)}_1$ $\overline{\texttt{(C⌋)}}$
2	comp-merge-predict	1	$\overline{\texttt{(N⌋)}}$ $\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌋)}}$
3	scan	2	$\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌋)}}$
4	spec-move-connect	2	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(+wh C⌈, -wh)}}$
5	shift	3	$\texttt{(=V +wh C⌋)}_1$ $_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(+wh C⌈, -wh)}}$
6	comp-merge-connect	3	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(V⌈, -wh)}}$
7	shift	4	$\texttt{(=N D⌋)}_1$ $_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(V⌈, -wh)}}$
8	comp-merge-predict	4	$\overline{\texttt{(N⌋)}}$ $\texttt{(D⌋)}_0$ $_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(V⌈, -wh)}}$
9	scan	5	$\texttt{(D⌋)}_0$ $_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(V⌈, -wh)}}$
10	spec-merge-connect	5	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(=D V⌈, -wh)}_0}$
11	nonfinal-merge-connect	5	$\overline{\texttt{(=D =D V⌋)}}$
12	scan	6	ϵ

Table 5: Parsing the object-movement derivation in Figure 4.

that" the predicted (C) constituent will be formed by a movement step which checks the -wh feature on the bottom-up recognized (D -wh) phrase. (Using spec-mv-connect at this point would not be an option if the bottom-up recognized category were simply (D).) The predicted (C) is therefore replaced with a prediction of category (+wh C,-wh), as we might expect: this is our prediction of a "sentence with a gap".

Of course we are not yet finished with the wh-phrase 'which wine'. It has two requirements: it must be moved into a position where its -wh feature can be checked and merged into a position where its D feature can be checked, and we have so far satisfied only the first of these. Before this spec-mv-connect step, both of these features were unchecked, as indicated by the placement of the "hook" in (D -wh⌋); this step has the effect of moving the hook leftward past the now-checked -wh feature, hence (D⌋-wh). The top of the hook "points" towards the remaining unchecked features and (as the parser progresses) moves in that direction, one step at a time, as features are checked. So (D -wh⌋) indicates that -wh is the next feature looking to be checked, hence the applicability of spec-mv-connect in Step 4, and (D⌋-wh) indicates that D is the next feature to be checked. Deciding when to check this feature amounts to deciding where to posit a gap site for the now-consumed filler. So although we do now have an obligation to find a place where this D feature can be checked somewhere within the (+wh C,-wh) that has now been predicted, the relationship between this bottom-up unchecked D and this top-down prediction is not the same as, for example, the relationship between the two symbols in the store at Step 5 of Table 4 (after consuming 'the queen'):

$$(\mathtt{D}\rceil)_0 \quad \overline{(\mathtt{V}\lceil)}$$

where the linear order in which words have been consumed constrains the positions within the predicted (V) where the D feature may be checked (i.e. it must be some position at the left edge, linearly). Instead, in the present case, we must maintain the obligation to check the remaining D feature somewhere within the (+wh C,-wh), but in a way that gives us free choice regarding when to do so. To record elements with this distinguished status — which are the same as the elements that have some but not all of their features currently checked — I use pre-subscripts, as shown after Step 4 in Table 5. I will sometimes refer to these subscripted categories as *bystanders*. Graphically, the intuition is that this category is "there if we want it", but we can also choose to "look over it" and work directly with the (+wh C,-wh) if we wish.

In Steps 5–10, we do exactly that. The complementizer is shifted (Step 5) and connected to the predicted (+wh C,-wh) (Step 6), and the bystander is passed down to the new predicted (V,-wh). In Steps 7–9 the subject is recognized, and it is then connected (Step 10) to the pending prediction to yield the new prediction of a (=D V,-wh), again with the bystander passed down. The store now looks like this:

$${}_{(\mathtt{D}\rceil\text{-}\mathtt{wh})_0}\overline{(\mathtt{=D\ V}\lceil\mathtt{,\ -wh})_0}$$

which we can read as saying that we need to fulfill a prediction of a (=D V,-wh), and we have up our sleeve a (D -wh) whose D feature remains unchecked, to use when we wish.

What happens next is the crucial "retrieval" step (Step 11). It is based on the MCFG-style rule

$$(\mathtt{=D\ V,\text{-}wh}) \quad \rightarrow \quad (\mathtt{=D\ =D\ V}) \quad (\mathtt{D\ \text{-}wh})$$

which is an instance of the MG rule nonfinal-merge, because the constituent whose selectee is checked has licensee features (here, -wh). Since the left-hand side of this rule corresponds to the category we already have a prediction of, it will be a connect transition (rather than a predict transition). (Notice that the =D selector feature on the left-hand side, and the one that appears in the parser's store at the stage under consideration, is not the one being checked at the (nonfinal-)merge step being discussed.) We connect the daughter we already have — this is when we decide to stop looking

over it, and bring it back into the fold — to the predicted parent, and replace that prediction with a prediction of the other daughter category, (=D =D V). This corresponds to drawing the two lines that branch down from the node labeled (=D V,-wh) in Figure 4, providing the (D -wh) node with the second dependency that it needs (the other one having already been established earlier, at the step corresponding to drawing the dashed line). The new prediction is straightforwardly fulfilled by scanning 'prefers' in the final step.

A perhaps unusual point to note about the nonfinal-merge-connect step is that it treats (D -wh) as the "left" corner of the relevant MCFG-style rule shown just above. This is *not* another instance of a right-daughter in the derivation tree linearly preceding its sister due to its being a specifier, of the sort we saw above when the subject 'the queen' was connected to top-down predictions in each of the two traces shown above: note that we would usually talk of the "trace" or "gap site" being to the *right* of the verb 'prefers' in this sentence, because the vacated position is a complement position. The fact which makes the wh-phrase a left-corner in the relevant sense here, however, is the fact that it linearly precedes the verb, due to its having moved to the left periphery of the sentence: of the two derivational children of the (=D V,-wh) node in the derivation tree, the wh-phrase is the one that the parser encounters first. Since movement is always to the left, transitions based on nonfinal-merge and spec-merge will share the property that the selected element precedes the selector and will therefore act as the "trigger" for the rule,[4] whereas transitions based on comp-merge will have the selector as the trigger.

In broad terms, we now have a concrete version of the "active gap-filling" heuristic that humans seem to adopt: it is a heuristic that says to always try nonfinal-merge transitions in preference to other kinds of transitions, in much the same way that the "late closure" can be seen as a heuristic that says to always try shift transitions in preference to reduce transitions. The particulars of the nonfinal-merge-connect step just discussed, however, of course should be compared with findings from the experimental psycholinguistics literature; I turn to this in Section 5.

Let us turn now to an example where the parser encounters the same fronted filler ('which wine') but resolves the dependency with a gap in a different position. Figure 5 shows the derivation tree for 'which wine prefers the queen', with the gap in subject position instead of object position. It uses the same lexical items as the previous example.

A trace of the parser's steps on this sentence is shown in Table 6. As one might expect, the first three steps are identical to the previous object-gap example (Table 5): we recognize the filler wh-phrase, bringing us to the configuration

$$(\texttt{D -wh}\rceil)_0 \quad \overline{(\texttt{C}\lceil)}$$

after 'wine' is scanned. More interestingly perhaps, the next three steps are identical to the previous example as well: we connect this wh-phrase as the specifier of the root CP ("drawing the dotted line") (Step 4), consume the complementizer (Step 5) and connect it to the current top-down prediction (Step 6) leaving us with only the predicted (V,-wh). At this point the parser is committed to analyzing 'which wine' as the specifier of the root CP, and has remaining only the task of finding a V projection that contains somewhere inside it a place for the filler's remaining D selectee feature to be checked. Both the object-gap and subject-gap sentences include this same initial portion of work. This is the improvement upon the top-down parser that we set out to achieve.

In Table 6, the next step (Step 7) performs the "retrieval", using (the remaining unchecked D feature of) the (D -wh) filler in accord with the nonfinal-merge instance

$$(\texttt{V,-wh}) \quad \rightarrow \quad (\texttt{=D V}) \quad (\texttt{D -wh})$$

[4]This is effectively no longer true in situations involving remnant movement, which I consider in Section 4.

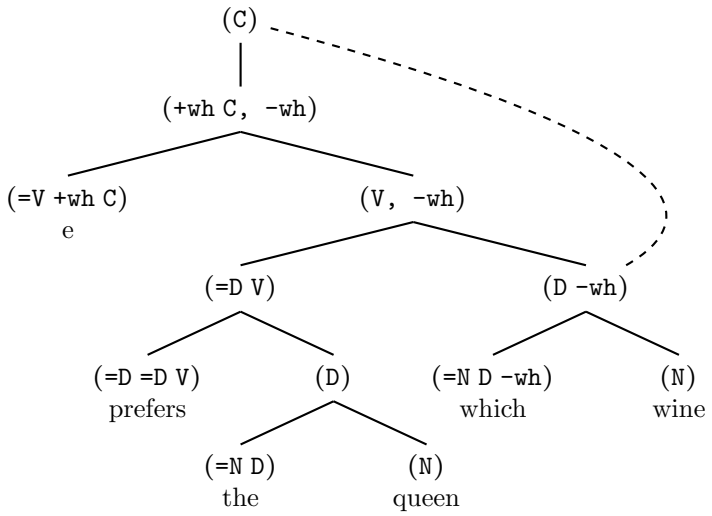


Figure 5: A derivation with one movement, from subject position. The parse is shown in Table 6.

to produce a new prediction of category (=D V). Fulfilling this prediction in the remaining steps involves no more movement, and is straightforward.

As in the previous example, the nonfinal-merge step (Step 7) corresponds to drawing the solid line that branches down to the (D -wh) node (and the one branching down to its sister). In the object-gap example, Step 7 (the first step that departs from what we see in Table 6) begins the process of consuming the subject 'the queen' (specifically, it shifts 'the'). So the decision the parser must make about which transition to take from the configuration shown in Step 6 in both tables is precisely the decision of whether or not to take the subject position to be the gap site.

In both of these two cases there is a transition based on a spec-move rule that analyses a wh-phrase as the specifier of a CP; and in particular, in both of these cases it analyses that wh-phrase as the specifier of the *root* CP, a constituent that we already have a prediction for. Things work in essentially the same way if this relevant CP constituent is *not* one that we are already expecting: the difference is simply the choice between spec-move-connect, which we have used in the two examples we have seen so far, and spec-move-predict. The effect of taking this "wrong turn" at Step 4 of Table 6 and having a fresh CP constituent be "brought into existence" in addition to the already predicted one (as one might need to do for an input like '[which wine prefers the queen] is an open question'), is shown in Table 7. Following the same sequence of steps thereafter, we end up stuck, having recognized the full utterance bottom-up and never connected it to the original prediction of the start category.

A schematic definition of the parsing rules is shown in Figure 6. An *item* is a category with or without a bar written over it, and with a "hook" marker indicating the division between checked and unchecked features; a *stack-element* is an item along with zero or more bystander items, written as left subscripts; the parser's store takes the form of a stack of stack-elements. Ellipses in the rules indicate that only the relevant top portion of the stack is shown explicitly. The shift and scan rules are also not shown here. Although not all of the ten rules shown here have been used in the examples above, they fill out a natural array of possibilities: for each of the five MG rules (three cases of merge, two cases of move), there is a predict rule and a connect rule, differing in whether the parent category is introduced as a fresh prediction (predict) or is taken to fulfill an existing

$_0$ which $_1$ wine $_2$ e $_3$ prefers $_4$ the $_5$ queen $_6$

Step	**Rule**	**Counter**	
0		0	$\overline{\texttt{(C⌋)}}$
1	shift	1	$\texttt{(=N D -wh⌋)}_1$ $\overline{\texttt{(C⌋)}}$
2	comp-merge-predict	1	$\overline{\texttt{(N⌋)}}$ $\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌋)}}$
3	scan	2	$\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌋)}}$
4	spec-move-connect	2	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(+wh C⌈, -wh)}}$
5	shift	3	$\texttt{(=V +wh C⌋)}_1$ $_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(+wh C⌈, -wh)}}$
6	comp-merge-connect	3	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(V⌈, -wh)}}$
7	nonfinal-merge-connect	3	$\overline{\texttt{(=D V⌈)}}$
8	shift	4	$\texttt{(=D =D V⌋)}_1$ $\overline{\texttt{(=D V⌈)}}$
9	comp-merge-connect	4	$\overline{\texttt{(D⌈)}}$
10	shift	5	$\texttt{(=N D⌋)}_1$ $\overline{\texttt{(D⌈)}}$
11	comp-merge-connect	5	$\overline{\texttt{(N⌈)}}$
12	scan	6	ϵ

Table 6: Parsing the subject-movement derivation in Figure 5.

$_0$ which $_1$ wine $_2$ e $_3$ prefers $_4$ the $_5$ queen $_6$

Step	**Rule**	**Counter**	
0		0	$\overline{\texttt{(C⌈)}}$
1	shift	1	$\texttt{(=N D -wh⌋)}_1$ $\overline{\texttt{(C⌈)}}$
2	comp-merge-predict	1	$\overline{\texttt{(N⌈)}}$ $\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌈)}}$
3	scan	2	$\texttt{(D -wh⌋)}_0$ $\overline{\texttt{(C⌈)}}$
4	spec-move-predict	2	$_{\texttt{(D⌋-wh)}_0}$ $\overline{\texttt{(+wh C⌈, -wh)}}$ $\texttt{(C⌋)}_0$ $\overline{\texttt{(C⌈)}}$
	...		...
11	comp-merge-connect	5	$\overline{\texttt{(N⌈)}}$ $\texttt{(C⌋)}_0$ $\overline{\texttt{(C⌈)}}$
12	scan	6	$\texttt{(C⌋)}_0$ $\overline{\texttt{(C⌈)}}$

Table 7: Illustrating spec-move-predict

Step	Rule	Remaining Input	
0		b e c d a	$\overline{\texttt{(E⌈)}}$
1	shift	e c d a	$\texttt{(B -f -g⌋)}_1 \quad \overline{\texttt{(E⌈)}}$
2	spec-move-connect	e c d a	$_{\texttt{(B -f⌉-g)}_1}\ \overline{\texttt{(+g E⌈, -g)}}$
3	shift	c d a	$\texttt{(=D +g E⌋)}_1 \quad _{\texttt{(B -f⌉-g)}_1}\ \overline{\texttt{(+g E⌈, -g)}}$
4	comp-merge-connect	c d a	$_{\texttt{(B -f⌉-g)}_1}\ \overline{\texttt{(D⌈, -g)}}$
5	nonfinal-move-connect	c d a	$_{\texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(+f D⌈, -f -g)}}$
6	shift	d a	$\texttt{(C -g⌋)}_1 \quad _{\texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(+f D⌈, -f -g)}}$
7	spec-move-connect	d a	$_{\texttt{(C⌉-g)}_1, \texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(+g +f D⌈, -g, -f -g)}}$
8	shift	a	$\texttt{(=A +g +f D⌋)}_1 \quad _{\texttt{(C⌉-g)}_1, \texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(+g +f D⌈, -g, -f -g)}}$
9	comp-merge-connect	a	$_{\texttt{(C⌉-g)}_1, \texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(A⌈, -g, -f -g)}}$
10	nonfinal-merge-connect	a	$_{\texttt{(B⌉-f -g)}_1}\ \overline{\texttt{(=C A⌈, -f -g)}}$
11	nonfinal-merge-connect	a	$\overline{\texttt{(=B =C A⌈)}}$
12	scan	ϵ	ϵ

Table 8: Parsing the multiple-movements derivation in Figure 7.

prediction (connect). The five MG rules are shown in a tree-based notation on the left, including quasi-daughters for the move rules to bring out the way (the relevant licensee features of) those quasi-daughters play the role of left-corners in the corresponding parsing rules.

3.3 Multiple movements

The examples shown so far involve only one movement step. Naturally, there are a variety of potential complications that arise in cases involving more than one movement step. The system as presented here handles some of these complications, but not all of them.

One variation which can be handled straightforwardly is multiple movements of a single constituent. For example, consider a derivation like the one in Figure 4 where the wh-phrase moves first to a specifier of the V-projection before going to its surface position. The original steps involved in attaching the filler to its surface position are unchanged. What happens at the intermediate position is something analogous to the "retrieval" steps so far — i.e. establishing a dependency in a nonfinal position — but rather than a nonfinal-merge transition to check a mover's base selectee feature as in Table 5 and Table 6, this intermediate retrieval is a nonfinal-move transition to check its intermediate licensee feature. A relevant (abstract) example, for the derivation in Figure 7, is shown in Table 8. The nonfinal-move transition is in Step 5, where the bystander's -f feature is checked.

Another variation which can be dealt with using the ideas illustrated so far is the "smuggling" configuration, where one phrase moves and a subconstituent of that phrase moves to a yet higher position. An example of a successfully parsed smuggling configuration is shown in Table 9, for the derivation in Figure 8. This derivation involves fronting of the A-projection, whose yield when it moves is 'a b', to check a -f feature; and then subsequent movement of the contained B-projection to a higher position to check a -g feature. This fronted B-projection is recognized bottom-up and then connected as the highest specifier of the root C-projection (Step 2) in the now-familiar manner.

Grammar Rule		Predict Rule
c-mrg	(α,ψ) $(\texttt{=f}\alpha)_1 \quad (\texttt{f},\psi)$	$(\texttt{=f}\alpha\rceil)_1 \quad \cdots$ $\vdash \overline{(\texttt{f}\lceil,\psi)} \quad (\alpha\rceil,\psi)_0 \quad \cdots$
s-mrg	$(\alpha,\phi\psi)$ $(\texttt{=f}\alpha,\phi)_0 \quad (\texttt{f},\psi)$	$(\texttt{f}\rceil,\psi) \quad \cdots$ $\vdash \overline{(\texttt{=f}\alpha\lceil,\phi)_0} \quad (\alpha\rceil,\phi\psi)_0 \quad \cdots$
n-mrg	$(\alpha,\phi\gamma\psi)$ $(\texttt{=f}\alpha,\phi) \quad (\texttt{f}\gamma,\psi)$	${}_{(\texttt{f}\rceil\gamma,\psi)\uplus\mathcal{B}_1\uplus\mathcal{B}_2}\overline{(\mathbf{I})} \quad \cdots$ $\vdash {}_{\mathcal{B}_1}\overline{(\texttt{=f}\alpha\lceil,\phi)} \quad (\alpha\rceil,\phi\gamma\psi)_0 \quad {}_{\mathcal{B}_2}\overline{(\mathbf{I})} \quad \cdots$
s-mv	$(\alpha,\phi\psi)$ $(\texttt{+f}\alpha,\phi\texttt{-f}\psi) \quad (\beta\texttt{-f},\chi)$	$(\beta\texttt{-f}\rceil,\chi) \quad \cdots$ $\vdash {}_{(\beta\rceil\texttt{-f},\chi)}\overline{(\texttt{+f}\alpha\lceil,\phi\texttt{-f}\psi)} \quad (\alpha\rceil,\phi\psi)_0 \quad \cdots$
n-mv	$(\alpha,\phi\gamma\psi)$ $(\texttt{+f}\alpha,\phi(\texttt{-f}\gamma)\psi) \quad (\beta\texttt{-f}\gamma,\chi)$	${}_{(\beta\texttt{-f}\rceil\gamma,\chi)\uplus\mathcal{B}_1\uplus\mathcal{B}_2}\overline{(\mathbf{I})} \quad \cdots$ $\vdash {}_{(\beta\rceil\texttt{-f}\gamma,\chi)\uplus\mathcal{B}_1}\overline{(\texttt{+f}\alpha\lceil,\phi(\texttt{-f}\gamma)\psi)} \quad (\alpha\rceil,\phi\gamma\psi)_0 \quad {}_{\mathcal{B}_2}\overline{(\mathbf{I})} \quad \cdots$

Grammar Rule		Connect Rule
c-mrg	(α,ψ) $(\texttt{=f}\alpha)_1 \quad (\texttt{f},\psi)$	$(\texttt{=f}\alpha\rceil)_1 \quad {}_{\mathcal{B}}\overline{(\alpha\lceil,\psi)} \quad \cdots$ $\vdash {}_{\mathcal{B}}\overline{(\texttt{f}\lceil,\psi)} \quad \cdots$
s-mrg	$(\alpha,\phi\psi)$ $(\texttt{=f}\alpha,\phi)_0 \quad (\texttt{f},\psi)$	$(\texttt{f}\rceil,\psi) \quad {}_{\mathcal{B}}\overline{(\alpha\lceil,\phi\psi)} \quad \cdots$ $\vdash {}_{\mathcal{B}}\overline{(\texttt{=f}\alpha\lceil,\phi)_0} \quad \cdots$
n-mrg	$(\alpha,\phi\gamma\psi)$ $(\texttt{=f}\alpha,\phi) \quad (\texttt{f}\gamma,\psi)$	${}_{(\texttt{f}\rceil\gamma,\psi)\uplus\mathcal{B}_1\uplus\mathcal{B}_2}\overline{(\alpha\lceil,\phi\gamma\psi)} \quad \cdots$ $\vdash {}_{\mathcal{B}_1\uplus\mathcal{B}_2}\overline{(\texttt{=f}\alpha\lceil,\phi)} \quad \cdots$
s-mv	$(\alpha,\phi\psi)$ $(\texttt{+f}\alpha,\phi\texttt{-f}\psi) \quad (\beta\texttt{-f},\chi)$	$(\beta\texttt{-f}\rceil,\chi) \quad {}_{\mathcal{B}}\overline{(\alpha\lceil,\phi\psi)} \quad \cdots$ $\vdash {}_{(\beta\rceil\texttt{-f},\chi)\uplus\mathcal{B}}\overline{(\texttt{+f}\alpha\lceil,\phi\texttt{-f}\psi)} \quad \cdots$
n-mv	$(\alpha,\phi\gamma\psi)$ $(\texttt{+f}\alpha,\phi(\texttt{-f}\gamma)\psi) \quad (\beta\texttt{-f}\gamma,\chi)$	${}_{(\beta\texttt{-f}\rceil\gamma,\chi)\uplus\mathcal{B}}\overline{(\alpha\lceil,\phi\gamma\psi)} \quad \cdots$ $\vdash {}_{(\beta\rceil\texttt{-f}\gamma,\chi)\uplus\mathcal{B}}\overline{(\texttt{+f}\alpha\lceil,\phi(\texttt{-f}\gamma)\psi)} \quad \cdots$

Figure 6: Much of the notation here is carried over from Figure 1. I write $\uplus$ for disjoint union, and $x \uplus S$ as a shorthand for $\{x\} \uplus S$; $(\mathbf{I})$ ranges over items (with the parentheses just for visual consistency); $\mathcal{B}, \mathcal{B}_1, \mathcal{B}_2, \ldots$ are variables ranging over sets of items ("bystanders"). A subscript 1 indicates lexical; a subscript 0 indicates non-lexical; absence of a subscript indicates unspecified lexicality.

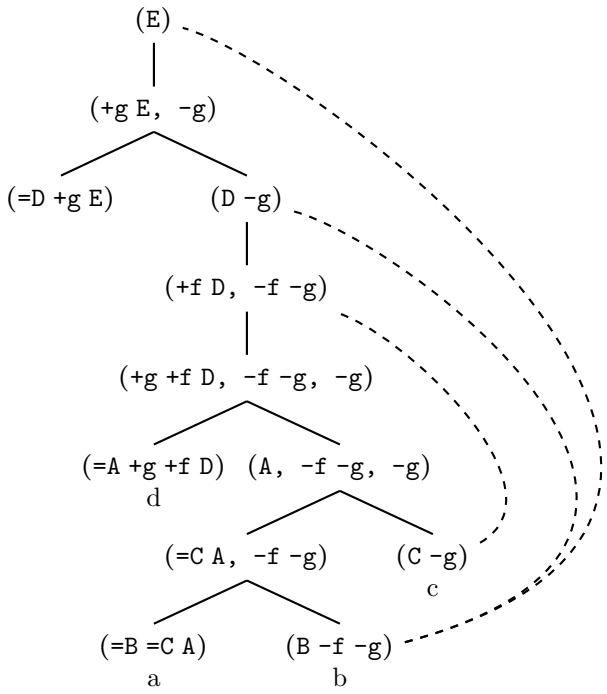


Figure 7: A derivation with multiple movements. The parse is shown in Table 8.

What is distinctive is the next step, Step 3: based on the nonfinal-merge rule

$$\texttt{(A -f,-g)} \quad \rightarrow \quad \texttt{(=B A -f)} \quad \texttt{(B -g)}$$

the bystander's outstanding `B` feature is used to predict the `A`-projection that it has moved out of. This leaves the predicted `(+g C,-g)` unchanged, but introduces other "immediate" (i.e. top of the stack) work to be done, namely fulfilling the newly-added prediction of category `(=B A -f)` which is achieved by scanning 'a' (Step 4), at which time we have now recognized the full `A`-projection bottom-up. This can now be connected as a lower specifier of the `C`-projection in Step 5, just as the `B`-projection was connected as its higher specifier earlier in Step 2.

Unfortunately, remnant movement — the "reverse" of a smuggling configuration, where a phrase out of which movement has already taken place itself moves — cannot be handled by the system introduced in this section. I introduce one somewhat tentative extension in the next section which attempts to address this issue.

4 Dealing with remnant movement

I will first, in Section 4.1, try to pinpoint the complications posed by these configurations and build up some intuitions for one way to attempt to tackle these problems, and then in Section 4.2 I will present one implementation of these ideas that correctly handles at least some cases of remnant movement.[5]

[5]It is probably also useful to note that the system presented in the previous section seems closely related to the "MGs with hypothetical reasoning" system in Kobele (2010), which corresponds exactly to MGs with remnant movement disallowed. Specifically, the spec-move transitions here seem to play the role of the discharge rule in Kobele's system,

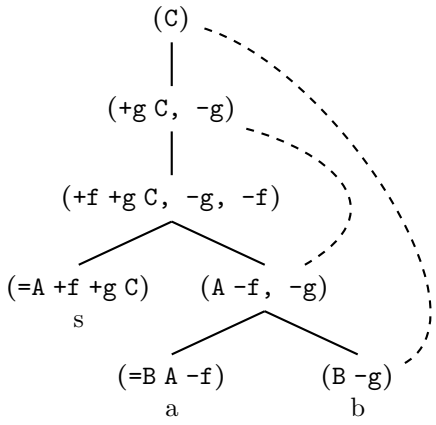


Figure 8: A smuggling derivation. The parse is shown in Table 9.

Step	**Remaining Input**		
0		b a s	$\overline{\texttt{(C}\rceil\texttt{)}}$
1	shift	a s	$\texttt{(B -g}\rceil\texttt{)}_1$ $\overline{\texttt{(C}\rceil\texttt{)}}$
2	spec-move-connect	a s	$_{\texttt{(B}\rceil\texttt{-g)}_1}$ $\overline{\texttt{(+g C}\lceil\texttt{, -g)}}$
3	nonfinal-merge-predict	a s	$\overline{\texttt{(=B A -f}\rceil\texttt{)}}$ $\texttt{(A -f}\rceil\texttt{, -g)}_0$ $\overline{\texttt{(+g C}\lceil\texttt{, -g)}}$
4	scan	s	$\texttt{(A -f}\rceil\texttt{, -g)}_0$ $\overline{\texttt{(+g C}\lceil\texttt{, -g)}}$
5	spec-move-connect	s	$_{\texttt{(A}\rceil\texttt{-f, -g)}_0}$ $\overline{\texttt{(+f +g C}\lceil\texttt{, -f, -g)}}$
6	nonfinal-merge-connect	s	$\overline{\texttt{(=A +f +g C}\rceil\texttt{)}}$
7	scan	ϵ	ϵ

Table 9: Parsing the smuggling derivation in Figure 8.

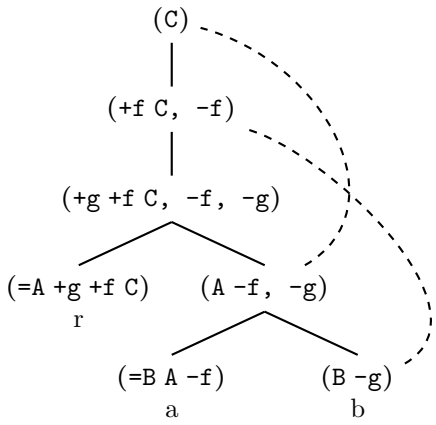


Figure 9: A remnant movement derivation.

4.1 An outline of the challenge

To illustrate, consider the derivation in Figure 9. This is minimally different from the smuggling example above, differing just in the order in which the two movements are triggered: here the `+f` licensor, which attracts the larger `A`-projection, is checked in the higher specifier of `C`, and the `+g` licensor, which attracts the smaller `B`-projection, is checked in the lower specifier of `C`. (The lexical entries where these orders of licensors are specified have pronunciations 's' for "smuggling" and 'r' for "remnant movement".) The derived word order here is therefore 'a b r'.

The first symbol in the input is 'a', so we will reach

$$(\texttt{=B A -f}\rceil)_1 \quad \overline{(\texttt{C}\rceil)}$$

after an initial shift step. Following the logic we have been using so far, it seems like the relevant next step should be a predict step on the basis of the nonfinal-merge rule

$$(\texttt{A -f,-g}) \quad \rightarrow \quad (\texttt{=B A -f}) \quad (\texttt{B -g})$$

which we would expect to result in replacing $(\texttt{=B A -f}\rceil)_1$ in our store with the two elements

$$\overline{(\texttt{B}\lceil\texttt{-g})} \qquad \text{and} \qquad (\texttt{A -f}\rceil\texttt{,-g})_0$$

in some configuration or other. That is to say, we expect the new state to be somehow constructed out of the following three elements.

- $\overline{(\texttt{C}\rceil)}$ the "underlying prediction"
- $\overline{(\texttt{B}\lceil\texttt{-g})}$ the "extractee"
- $(\texttt{A -f}\rceil\texttt{,-g})_0$ the "remnant"

the nonfinal-merge transitions play the role of assume rule, and bystanders play the role of hypothesized constituents (out of which no movement is allowed). (It is possibly natural that a parser might end up emulating something close to this assume/discharge system because the derivation trees in this system put lexical items in essentially their surface word order.) So the task of working out how to enrich the parsing system presented here to deal with remnant movement might be helped by considering the difference between the assume/discharge system and the full MG formalism.

Notice that the extractee here is the first instance we have encountered of a top-down predicted category with multiple "negative polarity" features, i.e. both a selectee `B` and a licensee `-g`. The challenge posed by remnant movement appears to be dealing with these kinds of elements, in particular understanding what status they have when they have had some, but not all, of those negative polarity features checked as the extractee has here. (The cases of partially-checked negative polarity features that we *have* seen involve constituents that have been found bottom up and then had a licensee feature checked in a post-movement position, with a selectee feature remaining unchecked.)

To try to isolate out the difficulty that these configurations raise, let us consider how we expect the underlying prediction and the remnant to interact as the parser continues its work, leaving aside the extractee temporarily. The next thing that one would expect to happen would be a spec-move-connect step that puts the remnant in the highest specifier position of the `C`-projection, of the sort that we have seen many times now. Keeping attention restricted to these two elements (i.e. abstracting away from the fact that the remnant has not, in fact, been completely recognized), we would expect such a transition to look something like this:

(13)

		b r	$\texttt{(A -f}\rceil\texttt{, -g)}_0$ $\overline{\texttt{(C}\rceil\texttt{)}}$
	spec-move-connect	b r	$_{\texttt{(A}\rceil\texttt{-f, -g)}_0}$ $\overline{\texttt{(+f C}\lceil\texttt{, -f)}}$

As written, this describes the standard situation where what remains is to find some (phonetically null) base position in which the bystander's `A` feature can be checked, as we work towards fulfilling the prediction of a `(+f C,-f)`. (The presence of the `-g` mover as part of this bystander category is of course related to the issues that we are temporarily abstracting away from.) At some future point we expect a nonfinal-merge transition to "retrieve" this bystander in the now-familiar manner. Specifically, we expect this to happen when we have worked our way down to a prediction of a `(+g +f C,-f,-g)`, since this is the mother of `(A -f,-g)` in the derivation tree. Note that by the time this happens, we expect that the extractee 'b' has been somehow consumed, since its surface position is to the left of the base position of the `A`-projection. So we expect the final steps of the parser's actions to be as follows:

(14)

		r	$_{\texttt{(A}\rceil\texttt{-f, -g)}_0}$ $\overline{\texttt{(+g +f C}\lceil\texttt{, -f, -g)}}$
	nonfinal-merge-connect	r	$\overline{\texttt{(=A +g +f C}\rceil\texttt{)}}$
	scan	ϵ	ϵ

Note that in this simple, minimal example, "all that happens" between (13) and (14) is the critical (and as yet somewhat mysterious) recognition of the extractee 'b' in its base position (where it checks `-g`): this is what allows the predicted `(+f C,-f)` in (13) to become `(+g +f C,-f,-g)` in (14). In remnant movement configurations more generally, however, the state that here exists only while we have a prediction of category `(+f C,-f)`, with both movement dependencies "in progress", will persist over some larger portion of the derivation.

So the question we now wish to ask is what information about `(B -g)` needs to be added to what is shown in the store in (13), and how should this information bear on the parser's actions as it moves to the configuration shown at the beginning of (14)?

A crucial point to realize appears to be that the availability of the nonfinal-merge-connect step shown in (14), i.e. the "retrieval" of the remnant, should be contingent upon the fact that the extractee 'b' has already been consumed. This is indeed the case in (14). But if the string being encountered were, for example, 'a r b' (instead of 'a b r'), then the following sequence of actions should *not* be possible: process the 'a' as a fronted remnant out of which 'b' has moved, reaching the configuration shown in (13) but with 'r b' as the remaining input, and immediately apply the nonfinal-merge-connect transition shown in (14) ("retrieving" the `A`-projection and hypothesizing

that its gap position is here) and then scan the 'r' as the sister of this gap position. The reason this should not be allowed is that it has the consequence of hypothesizing that the surface position of the extractee 'b' is further to the right of the base position of the A-projection, since we have not seen 'b' by the time we posit the latter's gap; this is incompatible with the assumption we made by analyzing 'a' alone (not the string 'a b') as a fronted A-projection. Put differently, when we analyze 'a' alone as fronted remnant out of which 'b' has moved, we not only commit ourselves to finding a base position for the A-projection and a surface position for 'b' further downstream — we commit ourselves to finding the surface position for 'b' *before* we find the base position for the A-projection, because the 'b' was extracted (leftwards) out of the base position occurrence of the A-projection.[6]

This suggests an answer to the question of what information about (B -g) is missing from the configuration shown at the end of (13): the bystander $(\texttt{A -f}\rceil, \texttt{-g})_0$ must be somehow temporarily inaccessible, its use contingent upon fulfilling the remaining part of the $\overline{(\texttt{B}\lceil\texttt{-g})}$ prediction that we abstracted away from.[7] This suggests a general structure where each stack element is a *tree*, containing not only daughters of the root node as bystanders in the now-familiar sense, but also daughters of those bystanders encoding "conditions" that must be satisfied before those bystanders can be used: each condition is a condition upon the use of one particular bystander (hence the nested structure), and in general there may be more than one condition for any single bystander (e.g. if there are multiple extractees from a moved remnant).

4.2 Outline of a candidate solution

Incorporating tree structure into our stack elements in the sense just mentioned allows us to correctly handle the remnant movement derivation in Figure 9, plus at least some degree of "iterated remnant movement" of the sort that is necessary for MGs to define non-context-free languages such as the copy language $\{ww \mid w \in \{a, b\}^*\}$ (Kobele, 2010).

Three new parsing rules are needed to manipulate these more complex structures: two based on nonfinal-merge, and one based on spec-move. These are shown in Figure 10. These rules use essentially the same notational conventions as those presented earlier in Figure 6, with the modification that $\mathcal{B}$ now ranges over *sets of trees of items* (rather than simply sets of items). As presented in Section 3 the system effectively operates on trees that are limited in depth to include at most one level of children (i.e. bystanders); I tentatively take these rules to apply to generalized trees "as is", simply allowing additional surrounding structure to be "carried around" until it is acted upon by the rules in Figure 10.[8] I write $\mathbf{I}\big[\mathcal{B}\big]$ for the tree with root node $\mathbf{I}$ and with child subtrees $\mathcal{B}$.[9] Note that in the system presented in Section 3, only top-down items had "bystander" children, and those children were always bottom-up elements. In the extended version here, bottom-up items can have children and those children will always be top-down elements. So although the depth of the trees is unbounded, given any two nodes that stand in an immediate dominance relationship there will

[6] We can imagine more complicated situations involving multiple interacting remnant movement configurations, where this does not hold. I will restrict attention here to situations where any lexical item has at most one licensee feature. Graf et al. (2016) showed that any MG can be converted into an equivalent one obeying this restriction.

[7] One might be tempted to suspect that the -g mover in the bystander category already encodes the relevant dependency, but this will not be sufficient in general: the phrase which is found in the surface -g position must be one that could have had a B feature checked inside the remnant. So if there were some other lexical item with category (D -g), then finding that item somewhere before we posit the gap position of the frontent A-projection remnant would not be sufficient.

[8] There may need to be some restrictions. For example, a bottom-up item with (top-down) bystanders, which will turn out to be what represents a fronted remnant with as-yet-unfound extractees, can be sensibly used as the left-corner trigger of a spec-move or nonfinal-move rule, but *not* as the left-corner trigger of a spec-merge rule, for example, because that is not compatible with this phrase moving. But it may turn out that "incorrectly" using such a phrase in a spec-merge rule would simply lead to dead-ends for the parser.

Grammar Rule		Secondary Predict Rule
n-mrg	$(\alpha, \phi\gamma\psi)$ $(\texttt{=f}\alpha, \phi) \quad (\texttt{f}\gamma, \psi)$	$\vdash \; {}_{\mathcal{B}}(\texttt{=f}\alpha\rceil, \phi) \quad \cdots$ $\overline{(\texttt{f}\lceil\gamma,\psi)}_{\cup\mathcal{B}}(\alpha\rceil, \phi\gamma\psi) \quad \cdots$

Grammar Rule		Secondary Connect Rule
n-mrg	$(\alpha, \phi\gamma\psi)$ $(\texttt{=f}\alpha, \phi) \quad (\texttt{f}\gamma, \psi)$	$\vdash \; {}_{\mathcal{B}_1}(\texttt{=f}\alpha\rceil, \phi) \quad {}_{\mathcal{B}_2}\overline{(\alpha\lceil, \phi\gamma\psi)} \quad {}_{\mathcal{B}_3}(\mathbf{I}) \quad \cdots$ $\overline{(\texttt{f}\lceil\gamma,\psi)}\big[\mathcal{B}_2\big]_{\cup\mathcal{B}_1\cup\mathcal{B}_3}(\mathbf{I}) \quad \cdots$

Grammar Rule		Secondary Rule
s-mv	$(\alpha, \phi\psi)$ $(\texttt{+f}\alpha, \phi\texttt{-f}\psi) \quad (\beta\texttt{-f}, \chi)$	$\vdash \; {}_{(\mathbf{I})}\big[\overline{(\beta\lceil\texttt{-f},\chi)}_{\cup\mathcal{B}_2}\big]_{\cup\mathcal{B}_1}\overline{(\alpha\lceil, \phi\psi)} \quad \cdots$ $\overline{(\beta\texttt{-f}\lceil, \chi)} \quad {}_{(\mathbf{I})}\big[\mathcal{B}_2\big]_{\cup\mathcal{B}_1}\overline{(\alpha\lceil, \phi\psi)} \quad \cdots$

Figure 10: Secondary parsing rules introduced for remnant movement.

always be one bottom-up element and one top-down element.

The two rules in Figure 10 based on nonfinal-merge allow for cases where the participating selectee occurs further to the right in the input than the selector — this is what occurs in remnant movement configurations, in contrast to the more straightforward cases of nonfinal-merge that are handled by the rules in Figure 6 where the selectee (having later moved leftwards) occurs further to the left. Specifically, these two new nonfinal-merge rules hypothesize that the $(\texttt{=f}\alpha, \phi)$ constituent that we have recognized is in fact part of a remnant-moved $(\alpha, \phi\gamma\psi)$ constituent; the rest of this constituent, i.e. $(\texttt{f}\gamma, \psi)$, must be predicted, of course, but we do not expect it to be found immediately. The status of such a constituent is what is encoded by being a top-down bystander. The two new nonfinal-merge rules produce such constituents, and stand to each other in the usual relationship that a predict/connect pair do. The new rule based on spec-move consumes, or retrieves, such top-down bystanders, and does not clearly fit the mould of either predict or connect rules.[10] I will refer to these three rules as "secondary rules", and their abbreviated names used in traces of the parser's actions will have the suffix '-sec'.

A trace of the parser's actions on the simple remnant movement derivation from Figure 9 is shown in Table 10. The first interesting point is the application of the secondary nonfinal-mrg-predict rule in Step 2. The effect is to hypothesize that the 'a' head that we have recognized is in fact all that we are going to see of the A-projection for now. This step therefore takes us into a configuration that has, essentially, a bottom-up recognized A-projection to be put towards fulfilling the underlying (C) prediction — *except that* we still have an obligation to find a (B -g) constituent elsewhere,

[9]The notation ${}_{\mathcal{B}}\mathbf{I}$ that has been used so far, is therefore equivalent to $\mathbf{I}\big[\mathcal{B}\big]$. I am introducing the latter only to avoid the need to iterate subscripts.

[10]This rule does form a natural class with the other two, in the sense that it deals with movements where a phrase appears to the right of its trace. In the case of the two new nonfinal-merge rules, this coincides with the fact that the selectee appears to the right of its selector (in contrast to the situations dealt with by the simpler nonfinal-merge rules in Section 3). But the situations to which the new spec-move rule is applicable to not involve a "non-canonical" linear ordering of the mover/licensee and its attractor/licensor.

Step	Rule	Input	
0		a b r	$\overline{\texttt{(C⌋)}}$
1	shift	b r	$\texttt{(=B A -f⌉)}_1 \quad \overline{\texttt{(C⌋)}}$
2	nonfinal-merge-predict-sec	b r	$\underset{\texttt{(B⌈-g)}}{\overline{}}\ \texttt{(A -f⌉, -g)}_0 \quad \overline{\texttt{(C⌋)}}$
3	spec-move-connect	b r	$\underset{\texttt{(A⌉-f, -g)}_0\left[\overline{\texttt{(B⌈-g)}}\right]}{\overline{}}\ \overline{\texttt{(+f C⌈, -f)}}$
4	spec-move-sec	b r	$\overline{\texttt{(B -g⌋)}} \quad \underset{\texttt{(A⌉-f, -g)}_0}{\overline{}}\ \overline{\texttt{(+g +f C⌈, -g, -f)}}$
5	scan	r	$\underset{\texttt{(A⌉-f, -g)}_0}{\overline{}}\ \overline{\texttt{(+g +f C⌈, -g, -f)}}$
6	nonfinal-merge-connect	r	$\overline{\texttt{(=A +g +f C⌋)}}$
7	scan	ϵ	ϵ

Table 10: Parsing the remnant movement derivation in Figure 9. The crucial remnant movement configuration is established by nonfinal-merge-predict-sec.

specifically in a position where its -g feature is checked. At the next step, the A-projection is connected as a specifier of the predicted C-projection. This is an instance of attaching a filler while keeping its gap position unresolved so the bottom-up (A -f, -g) constituent becomes a bystander of the updated top-down prediction, as we have seen many times before (cf. (13)). But in addition now, it brings along its own children which become "embedded" one level further in the tree structure. The resulting tree structure encodes the familiar relationship between (A -f, -g) and (+f C, -f) in the same way as before, but with the additional information that the base position of the (A -f, -g) projection can not be posited until the predicted (B -g) has been found (based on the logic outlined in the previous subsection). The secondary spec-mv rule, which applies next in Step 4, initiates the satisfaction of this additional requirement: the doubly-embedded (B -g) item is "retrieved" and becomes a top-level top-down prediction, ready to be fulfilled by scanning (at Step 5).[11] The applicability of the familiar nonfinal-merge-connect rule in Step 6, identifying the gap position for the fronted A-projection, is dependent on the fact that the (A -f, -g) item has no more children at this point.

This derivation involved an instance of the secondary nonfinal-merge-predict rule; specifically, this rule corresponds to the application of nonfinal-merge that establishes the relationship between the selecting (=B A -f) head and its evacuated (B -g) complement, and it is an instance of a predict step because the resulting (A -f, -g) parent node is created as a fresh prediction. To illustrate the secondary nonfinal-merge-connect rule, consider the slightly different derivation shown in Figure 11. The only difference is that the fronted A-projection contains a (D) specifier in addition to its evacuated (B -g) complement. A consequence of this is that after the shift step that consumes the 'a' (Step 3), there is a predicted A-projection already waiting on the stack, having been created when the initial (D) phrase was analyzed as a specifier (Step 2). This predicted A-projection is the parent (in the derivation tree) of the application of nonfinal-merge that establishes the relationship between the selecting (=B =D A -f) head and its evacuated (B -g) complement, and so the secondary nonfinal-merge-connect rule applies in Step 4. The configuration we reach is identical to the one reached in Table 10 after Step 2: an A-projection has been identified bottom-up, but with a gap inside it

[11] One can imagine instead supposing that this is achieved by shifting on the relevant lexical item, and then "matching" the resulting bottom-up item with the doubly-embedded (B -g) to cancel the latter out. This seems preferable in the sense that it would avoid spurious predictions about when this element is going to be encountered in the input. I'm opting for the other version only based on an intuition that we should stay in the style of an arc-eager left-corner parser (which the rest of the system has been based on), whereas the cancelling-out alternative seems more in line with the arc-standard versions.

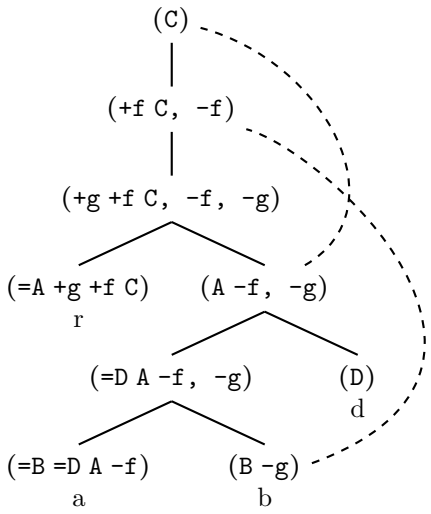


Figure 11: Another remnant movement derivation.

corresponding to the base position of a (B -g) constituent. In the earlier example in Table 10 this bottom-up A-projection was only brought into existence at the point where the remnant-movement configuration was created, so it was brought into existence with the top-down (B -g) prediction as a child[12]; the connect case in Table 11 is more complicated because the bottom-up A-projection is already there, so the prediction of the (B -g) constituent whose gap is posited by secondary nonfinal-merge-connect at Step 4 must be added to, or "passed up to", that existing A-projection.[13]

To illustrate that these new rules generalize at least somewhat beyond the specific cases used to motivate them here, Table 12 shows how they work on the complex remnant-movement derivation shown in Figure 12. This derivation corresponds to the derivation of the string 'aa' in the MG given in Stabler (2011) for the copy language $\{ww \mid w \in \{a, b\}^*\}$, but again writing 'e' for empty lexical items for clarity. Notice that when the secondary nonfinal-merge-connect rule applies in Step 4, the new top-down (A -r, -l) bystander inherits the bottom-up (T -r -l) bystander from the prediction that is being completed; since this filler's gap position had not been found in the course of fulfilling this (+l T -l, -r -l) prediction, its gap site must be in that part of the (+l T -l, -r -l) which was evacuated, i.e. the (A -r -l) constituent that is hypothesized at this step (as discussed in footnote 13). This means that when the (T -l, -r) constituent is connected as a specifier in Step 5, a tree structure with *three* levels of children is created. When the secondary specifier-move rule applies in Step 6 to "retrieve" the evacuated constituent's prediction, the entire tree structure whose root is the (A -r, -l) item is brought to the top of the stack; from this point on the top-down predictions with bottom-up bystanders behave precisely in accord with the simpler system laid out in Section 3.

While I take it to be promising that the rules introduced in this subsection naturally extended to this unintuitive case without ad hoc modifications, a proper investigation of the limits of what they

[12]More generally, with reference to Figure 10: the bottom-up $(\alpha, \phi\gamma\psi)$ item created by the secondary nonfinal-merge-predict rule gets this new top-down $(\mathtt{f}\gamma, \psi)$ bystander *in addition to* any analogous top-down bystanders (denoted by $\mathcal{B}$ in Figure 10) that had already been accumulated within the bottom-up $(\mathtt{=f}\alpha, \phi)$ item.

[13]More generally, with reference to Figure 10: the existing bottom-up item not only gets the new top-down $(\mathtt{f}\gamma, \psi)$ bystander, but also inherits any analogous top-down bystanders ($\mathcal{B}_1$) that had been accumulated within the bottom-up $(\mathtt{=f}\alpha, \phi)$ item. In addition, any bottom-up bystanders (i.e. already-processed fillers) for which a gap position was still to be found inside the predicted $(\alpha, \phi\gamma\psi)$ constituent must have their gap position inside the evacuated $(\mathtt{f}\gamma, \psi)$ subconstituent, so those are passed down as embedded bystanders ($\mathcal{B}_2$) for the new top-down item.

Step	Rule	Input	
0		d a b r	$\overline{\texttt{(C⌋)}}$
1	shift	a b r	$\texttt{(D⌋)}_1$ $\overline{\texttt{(C⌋)}}$
2	spec-merge-predict	a b r	$\overline{\texttt{(=D A -f⌈, -g)}_0}$ $\texttt{(A -f⌉, -g)}_0$ $\overline{\texttt{(C⌋)}}$
3	shift	b r	$\texttt{(=B =D A -f⌋)}_1$ $\overline{\texttt{(=D A -f⌈, -g)}_0}$ $\texttt{(A -f⌉, -g)}_0$ $\overline{\texttt{(C⌋)}}$
4	nonfinal-merge-connect-sec	b r	$\overline{\texttt{(B⌈-g)}}$ $\texttt{(A -f⌉, -g)}_0$ $\overline{\texttt{(C⌋)}}$
5	spec-move-connect	b r	$\texttt{(A⌉-f, -g)}_0\big[\overline{\texttt{(B⌈-g)}}\big]$ $\overline{\texttt{(+f C⌈, -f)}}$
6	spec-move-sec	b r	$\overline{\texttt{(B -g⌋)}}$ $\texttt{(A⌉-f, -g)}_0$ $\overline{\texttt{(+g +f C⌈, -g, -f)}}$
7	scan	r	$\texttt{(A⌉-f, -g)}_0$ $\overline{\texttt{(+g +f C⌈, -g, -f)}}$
8	nonfinal-merge-connect	r	$\overline{\texttt{(=A +g +f C⌋)}}$
9	scan	ϵ	ϵ

Table 11: Parsing the remnant movement derivation in Figure 11. The crucial remnant movement configuration is established by nonfinal-merge-connect-sec this time.

can achieve is an ongoing task.

5 Empirical connections based on what we have so far

5.1 Active gap-filling

As mentioned above, to a good first approximation we can say that the active gap-filling strategy amounts to preferring transitions based on nonfinal-merge in this left-corner MG parser over others. This makes it possible for active gap-filling to be understood as one aspect of a strategy for searching through a search space, much as Late Closure can be understood as a preference for shift transitions over reduce transitions in a bottom-up parser.

Recall that the nonfinal-merge transitions in the examples above occur perhaps surprisingly early. Specifically, in a sentence with a gap in object position (Table 5), this transition occurs before the verb is consumed. This was in some ways an arbitrary choice, taking the already-consumed filler to in effect be the left-corner that "triggers" the relevant VP-forming rule. An alternative that is perhaps equally reasonable would be to wait until the verb is consumed, use *that* as the trigger for the VP rule, and add a top-down prediction of an object that is in turn fulfilled by the posited gap. Formally speaking, the question concerns exactly how we should generalize familiar notions such as top-down, bottom-up and left-corner to a setting where categories have their features checked at different points in time.

Interestingly, however, the choice between the two options just outlined corresponds fairly closely to the distinction Omaki et al. (2015) make between "hyper-active gap-filling" and "conservative active gap-filling". Omaki et al. present some evidence that the strategy humans adopt is in fact the former, more pro-active strategy, where an object-position gap is posited before the verb.

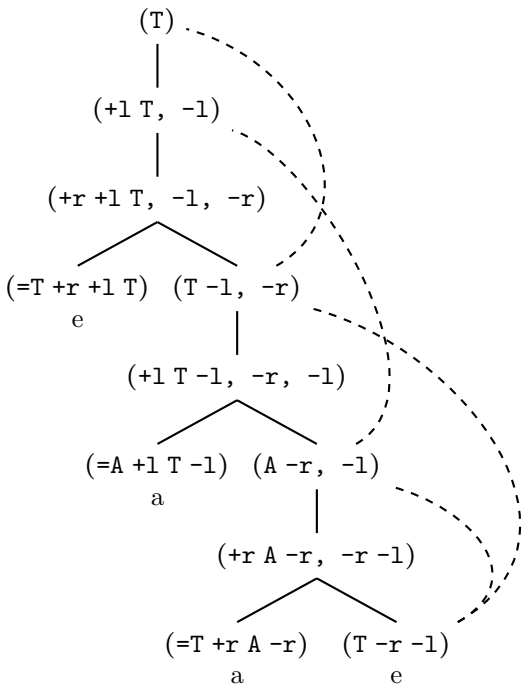


Figure 12: A complex remnant movement derivation, corresponding to a derivation from the copy language. The parse is in Table 12.

Step	**Rule**	**Input**	
0		e a a e	$\overline{\texttt{(T⌈)}}$
1	shift	a a e	$\texttt{(T -r -l⌋)}_1$ $\overline{\texttt{(T⌈)}}$
2	spec-move-predict	a a e	$_{\texttt{(T -r⌉-l)}_1}$ $\overline{\texttt{(+l T -l⌈, -r, -l)}}$ $\texttt{(T -l⌋, -r)}_0$ $\overline{\texttt{(T⌈)}}$
3	shift	a e	$\texttt{(=A +l T -l⌋)}_1$ $_{\texttt{(T -r⌉-l)}_1}$ $\overline{\texttt{(+l T -l⌈, -r, -l)}}$ $\texttt{(T -l⌋, -r)}_0$ $\overline{\texttt{(T⌈)}}$
4	nonfinal-merge-connect-sec	a e	$\overline{\texttt{(A⌈-r, -l)}}\left[_{\texttt{(T -r⌉-l)}_1}\right]$ $\texttt{(T -l⌋, -r)}_0$ $\overline{\texttt{(T⌈)}}$
5	spec-move-connect	a e	$_{\texttt{(T⌉-l, -r)}_0}\left[\overline{\texttt{(A⌈-r, -l)}}\left[_{\texttt{(T -r⌉-l)}_1}\right]\right]$ $\overline{\texttt{(+l T⌈, -l)}}$
6	spec-move-sec	a e	$_{\texttt{(T -r⌉-l)}_1}$ $\overline{\texttt{(A -r⌈, -l)}}$ $_{\texttt{(T⌉-l, -r)}_0}$ $\overline{\texttt{(+r +l T⌈, -r, -l)}}$
7	nonfinal-move-connect	a e	$_{\texttt{(T⌉-r -l)}_1}$ $\overline{\texttt{(+r A -r⌈, -r -l)}}$ $_{\texttt{(T⌉-l, -r)}_0}$ $\overline{\texttt{(+r +l T⌈, -r, -l)}}$
8	nonfinal-merge-connect	a e	$\overline{\texttt{(=T +r A -r⌈)}}$ $_{\texttt{(T⌉-l, -r)}_0}$ $\overline{\texttt{(+r +l T⌈, -r, -l)}}$
9	scan	e	$_{\texttt{(T⌉-l, -r)}_0}$ $\overline{\texttt{(+r +l T⌈, -r, -l)}}$
10	nonfinal-merge-connect	e	$\overline{\texttt{(=T +r +l T⌈)}}$
11	scan	ϵ	ϵ

Table 12: Parsing the complex remnant movement derivation in Figure 12.

5.2 Island sensitivity

Recall from Section 1 that active gap-filling appears to be island-sensitive: evidence suggests that readers do not posit a gap in the object position of 'wrote' in (3). While a satisfying explanation of island constraints remains elusive, it is not difficult to encode the standard generalizations in the MG formalism: for example, we can disallow extraction from within specifiers by disallowing spec-merge in cases where the specifier-to-be is a non-singleton category (i.e. has moving sub-parts).

Given a grammar that enforces island constraints in this way, it may be tempting to suspect that effects such as the absence of a posited gap in the object position of 'wrote' in (3) would follow immediately, since the hypothesis space being searched by the parser is precisely the space of possibilities defined by the grammar. (This was my own suspicion before I started writing this paper.) But things are not quite this simple. It is true that, if a grammar disallows extraction from specifiers, then the parser working with that grammar will not construct filler-gap dependencies where a specifier dominates the gap but not the filler; but this does not yet say anything about *at which point* in the sentence an attempt to construct such a dependency will have to be abandoned.

For example, consider the illicit filler-gap dependency in (15). (This is a more useful example than (4), since relative clauses are islands irrespective of the position in which they occur.)

(15) * Who was [a friend of ____] arrested?

We are interested in the question of whether a gap is posited after reading the prefix 'Who was a friend of'. Notice that in such a situation, the subject which is in the midst of being processed would eventually play the role of a left corner that triggers a rule positing (say) a TP node with the subject as one daughter and a T′ node as the other. This is to say that the bracketed phrase "is not a specifier yet" at the point where the decision about whether to posit a gap after 'of' arises. So a grammatical prohibition on extraction out of specifiers will not prevent the positing of a gap in this position. What it *will* prevent is the use of the bracketed phrase *as a specifier*: at the position marked with a closing bracket, the parser will have recognized an NP with a filler-gap dependency reaching inside it, and will then move on to address the question of how to fit that NP into a larger structure. Only when it reaches this latter question will the specifier-island constraint come into play.

This clearly raises a number of subtle questions about the empirical landscape. At the very least, one would like to know whether there is evidence that readers do not posit island-violating gaps in situations analogous to (15) but with the gap site not at the right edge of the subject island. If humans' positing of gaps is indeed "fully island sensitive" in this manner, then this would suggest that the parser presented here has departed too far from Stabler's and is not top-down enough: in a fully top-down approach, the current phrase's position in the surrounding structure (e.g. whether it is a specifier or not) is always known.

6 Conclusion

I have presented an initial foray into the development of an incremental MG parser that would provide a formal grounding for the growing experimental literature on the formation of long-distance dependencies. The crucial idea is to allow the first link in a dependency (such as a wh-filler) to act as the left-corner which triggers a rule that introduces a prediction of a category that will contain the other link in the dependency (such as a gap). This contrasts with the top-down parser from Stabler (2013) which requires that all the links in a dependency by identified with particular structural positions before any of the words participating in the dependency are scanned. Remant

movement poses distinct challenges, due largely to the fact that they create situations where the parser encounters an element's trace before it encounters the element itself; I have outlined the beginnings of what may turn out to be a solution to these issues. The system does however already provide a solid handle on the robustly-attested active gap-filling strategy, which can be seen as a simple search heuristic, preferring a certain kind of transition through the search space over others.

References

Dillon, B. (2014). Syntactic memory in the comprehension of reflexive dependencies: An overview. *Language and Linguistic Compass*, 8(5):171–187.

Fodor, J. D. (1978). Parsing strategies and constraints on transformations. *Linguistic Inquiry*, 9(3):427–473.

Frazier, L. and Clifton, C. (1996). *Construal*. MIT Press, Cambridge, MA.

Graf, T., Aksënova, A., and De Santo, A. (2016). A single movement normal form for Minimalist grammars. In Foret, A., Morrill, G., Muskens, R., Osswald, R., and Pogodalla, S., editors, *Formal Grammar, FG 2016*, volume 9804 of *Lecture Notes in Computer Science*.

Kanazawa, M. (2016). Formal grammar: An introduction. Online lecture notes: http://research.nii.ac.jp/ kanazawa/FormalGrammar/index.html.

Kobele, G. M. (2010). Without remnant movement, MGs are context-free. In Ebert, C., Jäger, G., and Michaelis, J., editors, *Proceedings of Mathematics of Language 10/11*, volume 6149 of *LNCS*, pages 160–173, Berlin Heidelberg. Springer.

Michaelis, J. (2001). *On Formal Properties of Minimalist Grammars*. PhD thesis, Universität Potsdam.

Omaki, A., Lau, E. F., White, I. D., Dakan, M. L., Apple, A., and Phillips, C. (2015). Hyper-active gap filling. *Frontiers in Psychology*, 6(384).

Parker, D. and Phillips, C. (2016). Negative polarity illusions and the format of hierarchical encodings in memory. *Cognition*, 157:321–339.

Phillips, C. (2006). The real-time status of island phenomena. *Language*, 82:795–823.

Stabler, E. P. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *LNCS*, pages 68–95, Berlin Heidelberg. Springer.

Stabler, E. P. (2011). After Government and Binding theory. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*. Elsevier, second edition.

Stabler, E. P. (2013). Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science*, 5(3):611–633.

Stabler, E. P. and Keenan, E. L. (2003). Structural similarity within and among languages. *Theoretical Computer Science*, 293:345–363.

Stowe, L. A. (1986). Parsing wh-constructions: Evidence for on-line gap location. *Language and Cognitive Processes*, 1(3):227–245.

Traxler, M. J. and Pickering, M. J. (1996). Plausibility and the processing of unbounded dependencies: An eye-tracking study. *Journal of Memory and Language*, 35:454–475.

Wagers, M. W. and Phillips, C. (2009). Multiple dependencies and the role of grammar in real-time comprehension. *Journal of Linguistics*, 45:395–433.