Cyclic scope and processing difficulty in a Minimalist parser*

Robert Pasternak
Leibniz-Center for General Linguistics (ZAS)
mail@robertpasternak.com

Thomas Graf
Stony Brook University
mail@thomasgraf.net

February 4, 2020

Abstract

A common view in the theoretical literature is that quantifier raising (QR) is a clause-bounded operation. But in a paper published in *Glossa*, Wurmbrand (2018) argues that (i) QR is not clause-bounded, and the apparent clause-boundedness of QR is due to the human parser's difficulty in processing extraclausal QR; and (ii) the relative difficulty of extraclausal QR depends on the size of the embedded clause from which QR takes place. She then proposes a theory of scope processing in which parsing LF movement is a costly operation for the human parser, which in conjunction with independently motivated assumptions about A′-movement generates the desired results. In this paper, we accept Wurmbrand's descriptive observations and proposed syntax but argue against her theory of scope processing, as it does not capture the relationship between LF and PF as seen in other scope processing phenomena. We offer an alternative metric of scope processing difficulty in a formal framework, using Minimalist Grammars (Stabler 1997) and their associated parsers. The new metric accounts for Wurmbrand's observations as well as those cases that are problematic for her account, and points the way toward an explanatory theory of scope processing.

1 Introduction

A basic question that often is (or should be) asked when presented with a sentence that seems ungrammatical, or that seems to lack an otherwise expected reading, is whether this apparently unavailable structure is prohibited by the grammar proper—that is, it is ruled out by the speaker's *competence*, in Chomsky's (1965) sense—or whether it is a matter of *performance*, meaning that it is allowed by the grammar, but excessively difficult for the human parser to process.

Perhaps the most famous case where the answer seems to be the latter is center-embedding (Chomsky & Miller 1963). The right-embedding structure in (1a), while a bit lengthy, is still relatively easy to process for the average listener. However, if the relative clauses are switched from passive to active voice, we end up with the center-embedding structure in (1b), which to the average listener sounds like practical gibberish.

*Comments welcome and much appreciated. For a PDF of the formal appendices, please contact the first author.

(1) a. The mouse that was eaten by the cat that was bitten by the dog that was owned by the barber liked cheese.

b. The mouse that the cat that the dog that the barber owned bit ate liked cheese.

Nonetheless, for a variety of reasons the broad consensus is that (1b) is, in fact, a grammatical sentence of English, but one that presents an overwhelming task for the human parser. In other words, the apparent unacceptability of (1b) is a matter of performance, rather than competence.

A growing body of work has focused on asking the same competence-vs.-performance question about another observation that has been made in the syntactic-semantic literature: namely, that quantifier raising (QR) is an apparently clause-bounded operation, so that a quantifier cannot take scope outside of the clause in which it is merged.[1] Consider the examples in (2), due to Fox (2000):

(2) Fox 2000, p. 62:

a. Someone said that every man is married to Sue.

b. Someone said that Sue is married to every man.

As Fox notes, it seems that each sentence in (2) only allows for a reading in which someone claims that Sue is polygamous, and not one in which a variety of people have made differing claims about who Sue is married to. The former reading is one in which *someone* scopes above *every man* ($\exists > \forall$), while the latter (absent) reading would be the result of *every man* scoping outside of its clause and above *someone* ($\forall > \exists$). The apparent absence of this second reading suggests that *every man* cannot undergo QR to a position outside of its embedded clause.

Notice that if QR is a form of A′-movement, as is commonly thought to be the case, the putative clause-boundedness of this operation is surprising, since this is not a general constraint on A′-movement. This is illustrated for *wh*-movement, the prototypical case of A′-movement, in (3):

(3) Who$_1$ did someone say [$_{\text{CP}}$ t_1 that Sue is married to t_1]?

Thus, if cyclic QR is truly prohibited by the grammar, then either QR must not be a form of A′-movement (Hornstein 1994, but see Kennedy 1997 and Wilder 1997), or some alternative mechanism must be posited that traps quantificational DPs inside the clauses in which they are merged.[2]

But as will be discussed in greater detail in §2, in a paper published in *Glossa*, Wurmbrand (2018) argues that (i) QR is in fact not clause-bounded, and looks like any other form of A′-movement as far as the grammar proper is concerned; (ii) the apparent clause-boundedness of QR is a matter of performance rather than competence, as extraclausal QR is more difficult to process than within-clause QR (cf. Syrett & Lidz 2011; Syrett 2015a,b); and (iii) the relative difficulty of

[1]Note that this restriction excludes indefinites, which are well-known to be able to scope (or "scope") not only out of embedded clauses, but also out of islands. However, this is generally thought to be due either to some alternative mechanism of scope-taking for indefinites (see, e.g., Reinhart 1997), or to a distinct, referential use of the indefinite that gives the appearance of wide scope (see, e.g., Fodor & Sag 1982, Kratzer 1998).

[2]As an example of the latter, Fox (2000) captures the apparent clause-boundedness of QR by means of his principle of *Scope Economy*, which prohibits semantically vacuous covert movement. Fox argues that given certain basic movement constraints, cyclic QR would require at least one semantically vacuous iteration of QR, which Scope Economy prohibits. (See Cecchetto 2004 for similar arguments.) Of course, if cyclic QR turns out to be possible then the tables are turned, and Scope Economy must be revised in order to permit extraclausal QR. One possibility: independent of the issue of cyclic QR, Anderson (2004) provides experimental evidence against a hardline competence-based view of Scope Economy and in favor of a performance-based approach, in which case semantically vacuous QR may be unproblematic as far as the grammar proper is concerned.

processing extraclausal QR correlates with the size of the embedded clause from which QR takes place. To account for (iii), Wurmbrand proposes that movements that impact LF are costly to process, so that a greater cost is assigned to sentences where a quantificational DP is interpreted at a greater distance—measured in terms of the number of iterations of movement—from its merge position. This principle, paired with certain independently motivated assumptions about the nature of clausal complementation and A′-movement, derives the right predictions with respect to comparative difficulty in the processing of extraclausal QR: put simply, scoping out of larger embedded clauses requires more iterations of movement, and is therefore more difficult to process.

Wurmbrand's proposal exhibits a variety of traits desirable for a metric of scope processing difficulty: most notably, it makes specific, formally defined, and empirically testable claims, and is built on theoretical principles that have been previously motivated on independent grounds. However, towards the end of §2 we will discuss certain data that present difficulties for Wurmbrand's processing metric. In short, the problem is that she predicts scope processing difficulty to be determined only by LF, since all that matters is the distance between where a DP is interpreted and where it is merged. But it seems that what the structure/derivation looks like on the PF side of things also matters. For example, as Wurmbrand herself notes, it appears that overt cyclic *wh*-movement is significantly more easily processed than cyclic QR, in spite of the fact that both involve operators being semantically interpreted at great distances from their merge positions. This contrast between *wh*-movement and QR indicates that LF-affecting movement is noticeably easier to parse when it also impacts PF, i.e., that overt scopal movement is more readily processed than covert movement. Second, experimental results from Lee (2009) suggest that when a quantificational DP undergoes overt movement, it is more easily interpreted in its post-movement position than in its pre-movement position, even though reconstruction would entail the DP being interpreted closer to its merge position. It thus appears that the comparative difficulty of computing a given scope configuration does not depend exclusively on the relative complexity of the LF side of the derivation or representation, as in Wurmbrand's analysis, but rather on the relationship between PF and LF, so that movement is processed more easily when it affects both PF and LF than when it affects only LF (QR) or PF (reconstruction).

Wurmbrand provides multiple suggestions for how a revised version of her metric might account for some of these observations. However, her suggestions are only sketched rather than fully formalized, meaning that they lack the strong predictive power of their pre-revision predecessor. Nonetheless, given the nature of the problematic evidence, one of Wurmbrand's suggestions seems especially promising: namely, that whether a given scope configuration for a given sentence is easier or harder to process depends on the existence and severity of any mismatch between PF and LF representations. In this paper we will offer a novel theory of scope processing difficulty that is very much in the spirit of this insight, with sufficient predictive power to account for all of the data in question. Moreover, in addition to our descriptive aim of characterizing which scope configurations are more difficult to process than others, we will hopefully point the way toward an explanation of why this should be the case, i.e., what it is about the nature of the human parser that makes such configurations more difficult to process.

The theory of scope processing difficulty presented in this paper is built on a top-down parser for Minimalist Grammars (MGs, Stabler 1997), formal grammars incorporating certain core features from Chomsky's (1995) Minimalist Program. Since the pioneering work of Stabler (2013) and Kobele et al. (2013), top-down MG parsers have been used to explain a variety of syntactic processing phenomena attested in the psycholinguistic literature, including the greater processing difficulty of center- vs. right-embedding (Kobele et al. 2013, Gerth 2015); the lesser processing difficulty of Dutch serial verb constructions like (4), in contrast to German center-embedding as in (5) (Kobele et al. 2013); cross-linguistic preferences in parsing subject vs. object relative clauses (Graf et al. 2015, 2017); the human parser's handling of stacked relative clauses in English and Chinese (Zhang 2017); the comparative processing of sentences with and without Heavy NP Shift (Liu 2018); dative DP attachment ambiguities in Korean (Lee 2019); and a variety of word order and relative clause processing facts in Italian (De Santo 2019).

(4) Dutch serial verb construction ($N_1N_2N_3V_1V_2V_3$):

dat	Jan	Piet	Marie	zag	laten	zwemmen
that	Jan	Peter	Mary	saw	let	swim

'that Jan saw Peter let Mary swim' (Kobele et al. 2013, p. 35)

(5) German center-embedding ($N_1N_2N_3V_3V_2V_1$):

daß	Hans	Peter	Marie	schwimmen	lassen	sah
that	Hans	Peter	Mary	swim	let	saw

'that Hans saw Peter let Mary swim' (Kobele et al. 2013, p. 34)

Up to this point such work has focused on cases where the sentences being compared differed in their overt structure. Thus, to our knowledge this paper is the first attempt to extend this productive research program to cases where the detectable differences lie only in the comparative (un)availability of certain semantic interpretations.

We begin in §3 by introducing MGs, starting with a simpler variant in which only PF representations can be built, and then extending it in a way that allows for the simultaneous generation of both PF trees and LF trees in the style of Heim & Kratzer (1998).[3] In §4 we introduce a simplified version of the top-down parser for our extended MGs. This gives us a formal grammar and parser that can both produce and parse scope ambiguities using syntactic representations that largely accord with assumptions in the theoretical literature. With the formal apparatus in place, in §5 we define our principle of scope processing difficulty, which we refer to as the SLD Principle—a name that will remain opaque until the principle is properly introduced. But in short, the idea is as follows. During the course of a parse, the extended MG parser makes predictions about the existence and nature of various syntactic constituents, as well as predictions about where in the tree those constituents sit, both at PF and at LF. But when a constituent occupies distinct locations at PF and at LF—that is, when QR or reconstruction has occurred—there will be a number of parse steps during which that constituent has been predicted to exist, but its location at either PF or LF is not yet known. This sort of situation is what the SLD Principle treats as costly, with parses being deemed more costly if there are more such constituents, or if those constituents go longer (in terms of number of parse steps) before they are assigned locations at both PF and LF. After introducing the SLD Principle, we show in the rest of §5 that this principle derives all of the correct scope processing results as discussed in §2. Finally, we offer some concluding remarks in §6.

[3] Ours is not the first version of MGs to include scope-altering LF movement operations—even the original formulation in Stabler 1997 allows for this possibility. However, our formulation differs from Stabler's in several respects. First, it allows for the possibility of PF-only movement—that is, reconstruction—which is required for many of the examples discussed in this paper. Second, it generates two separate (PF and LF) trees, unlike Stabler's, where phonetic and semantic material occupy the same tree. And third, the LF trees we generate bear a closer resemblance to those commonly seen in the semantic literature, including indexed traces and co-indexed lambda-abstracting nodes appearing below LF landing sites.

2 Cyclic QR and Wurmbrand's (2018) analysis

In this section we discuss the currently available evidence in favor of the existence of extraclausal QR, as well as the comparative difficulty of different types of extraclausal QR. We then go over Wurmbrand's (2018) proposal, including her theory of scope processing difficulty, according to which processing is more difficult when quantifiers take scope at a greater distance from their merge positions. We follow this up by discussing some cases in which overt movement seems to not only facilitate but actively encourage a wider-scope interpretation, an observation that is problematic for Wurmbrand's analysis. We then note an alternative way of looking at things in terms of PF-LF mismatch, which is only sketched and hinted at by Wurmbrand, but which we take to be a more promising route. The formally fleshed out proposal in the rest of this paper can be thought of as an analysis along precisely these lines.

2.1 Evidence for cyclic QR

The evidence put forth in the theoretical and experimental literature in favor of the existence and comparative difficulty of extraclausal QR can be grouped into two broad categories. The first are straightforward truth value judgments for sentences with quantificational DPs in embedded clauses. The second are cases of antecedent-contained deletion (ACD) in which quantificational DPs must scope out of embedded clauses. We will discuss these in turn.

2.1.1 Embedded quantificational DPs

Perhaps the clearest cases attesting to the existence of extraclausal QR are sentences in which DPs that appear in embedded clauses can take scope over DPs that are merged in higher clauses. For instance, Larson & May (1990) and Kennedy (1997) note that quantifiers can scope out of non-finite clauses and over matrix-clause quantifiers; Kennedy provides the examples in (6) (his (41–47)):

(6) Kennedy 1997, p. 674:
  a. At least two American tour groups expect to visit every European country this year.
  b. Some agency intends to send aid to every Bosnian city this year.
  c. At least four recreational vehicles tried to stop at most AAA approved campsites this year.
  d. Some congressional aide asked to see every report.
  e. More than two government officials are obliged to attend every state dinner.
  f. A representative of each of the warring parties is required to sign every document.
  g. At least one White House official is expected to attend most of the hearings.

Kennedy notes that (6a), for example, allows an interpretation in which the two (or more) tour groups can vary from country to country, as would be expected if *every European country* takes scope over *at least two American tour groups*. Parallel facts hold for the other examples as well.

While Larson & May (1990) and Kennedy (1997) provide evidence that QR can escape non-finite clauses, they (and others) nonetheless maintain that quantifiers *cannot* QR out of *finite* clauses, as evidenced by sentences like those in (2), repeated below:

(2) Fox 2000, p. 62:

a. Someone said that every man is married to Sue.

b. Someone said that Sue is married to every man.

However, Farkas & Giannakidou (1996) argue that, at least for a limited range of embedding verbs and syntactic/semantic configurations, quantifiers in finite embedded clauses can scope over quantifiers in matrix clauses. One such example can be seen in (7) (their (6)), where the helpful students can covary with the invited speakers. This judgment can be somewhat sharpened by inserting *different* between *a* and *student*.

(7) Farkas & Giannakidou 1996, p. 36:
A student made sure that every invited speaker had a ride.

The apparent takeaway from the theoretical literature is thus that extraclausal QR is in fact possible, though obtaining inverse scope readings through extraclausal QR seems to be more difficult than in monoclausal cases, and there are perhaps construction-dependent restrictions on extraclausal QR from finite clauses.

On the experimental end of things, work in this area is limited, and is at this point more suggestive than it is conclusive. Moulton (2007), using experimental methods adopted from seminal work by Anderson (2004), attempts to answer (I) whether inverse scope is easier within the same clause (e.g., (8a)) than across (infinitival) clausal boundaries, and (II) whether infinitival complements of restructuring verbs like *try* are more conducive to quantifier extraction than those of non-restructuring verbs like *decide* and *plan* ((8b) vs. (8c)):[4]

(8) a. A technician inspected every plane.
b. A technician tried to inspect every plane.
c. A technician decided to inspect every plane.

Restructuring is a phenomenon in which normally clause-bounded operations can seemingly escape out of certain infinitival clauses, but with particular restrictions on which clauses are transparent in this manner. Consider, for example, the well-known case of the so-called "long passive", illustrated in (9). In (9a), the internal argument *der Traktor* ('the tractor') of the embedded verb *reparieren* ('repair') appears with nominative case as the subject of the matrix clause, with the matrix verb *versuchen* ('try') passivized. In other words, as far as the passivization operation is concerned, *versuchen* and *der Traktor* are treated as if they are clausemates, with the passivization of *versuchen* leading to the promotion to subject of the internal argument *der Traktor*. However, while this is possible for restructuring verbs like *versuchen*, other verbs disallow the long passive, as illustrated in (9b) with the non-restructuring *planen* ('plan').

(9) a.

dass	der Traktor	zu	reparieren	versucht	wurde
that	the tractor.NOM	to	repair	tried	was

'that they tried to repair the tractor'[5] (German, Wurmbrand 2001, p. 19)

b.

* dass	der Traktor	zu	reparieren	geplannt	wurde
that	the tractor.NOM	to	repair	planned	was

'that they planned to repair the tractor' (German, Wurmbrand 2001, p. 57)

[4]As we will see in §2.2, adopting a binary distinction between "restructuring" and "non-restructuring" verbs paints a somewhat misleading picture. However, for the time being this is a harmless oversimplification.

A common approach to the syntax of restructuring, dating back at least to the work of Rizzi (1978), is to posit that at some point in the derivation, whatever boundary prevents operations like the long passive in (9b) is absent in cases like (9a). One way of achieving this, following the work of Wurmbrand (2001, 2014a, 2015), is to say that the relevant operational boundary is absent from the get-go: verbs differ in the size of infinitival complements that they can take, with the infinitival complements of restructuring verbs being smaller than those of non-restructuring verbs, i.e., lacking certain higher heads along the clausal spine. If the structural configuration whose presence creates the relevant domain boundary is among those that appear only in the complements of non-restructuring verbs, then this will generate the observed differences.

While some of Kennedy's (1997) examples in (6) suggest that inverse scope is possible with non-restructuring verbs, the informal observation underlying Moulton's interest in question (II) is that inverse scope seems easier with restructuring verbs. If true, this is an intriguing observation given Wurmbrand's clause-size theory of restructuring. As Moulton notes, a Wurmbrand-style analysis could lend itself to a compelling explanation for why such an observation should hold: the embedded DP is extracted from a less complex structure, and thus moves a shorter distance, when the infinitival clause it is moved from is the complement of a restructuring verb.

Turning to the results of Moulton's experiment, he finds that in contexts biasing for an inverse scope interpretation, there is a clear distinction in acceptability between within-clause and across-clause QR, with participants accepting within-clause inverse scope at a statistically significantly higher rate than cross-clausal inverse scope. With respect to a distinction in extraclausal QR between restructuring and non-restructuring complements, Moulton's results fall just shy of statistical significance, in spite of a notable numerical difference between the two, with restructuring verbs receiving inverse scope interpretations 61% of the time, in contrast to only 49% for non-restructuring verbs. The relevant results for Moulton's experiment can be seen in Figure 1.[6]

monoclausal	**restructuring**	**non-restructuring**
.71	.61	.49

Figure 1: Inverse scope response rates in Experiment 1 of Moulton 2007 (n = 36)

We can therefore conclude that monoclausal inverse scope (e.g., (8a)) is easier to process than inverse scope from infinitival clauses ((8b) and (8c)). We will also follow Wurmbrand in concluding—tentatively, since the results are only suggestive rather than statistically significant—that extraclausal QR from *try*-type infinitives is easier to process than from *decide*-type infinitives.

The other relevant study involving truth value/acceptability judgments with quantificational DPs in embedded clauses comes from Tanaka (2015a,b). Tanaka's main goal in her dissertation (Tanaka 2015b) is to explore parallels between *wh*-movement and QR in terms of processing difficulty from a variety of weak islands. While these results are interesting, of greater interest for our purposes is a follow-up study of hers focusing specifically on QR from finite embedded clauses.

[5]In her examples, Wurmbrand uses embedded clauses with the complementizer *dass* ('that'), rather than simple matrix clauses, in order to avoid any interference from verb-second effects in German.

[6]Moulton also tests whether there is a difference between "normal" restructuring verbs like *try* and implicative restructuring verbs like *manage*, but finds no significant difference between them: both are accepted 61% of the time. He additionally discusses a second experiment of his attempting to further differentiate between infinitival complements; none of the results there achieve significance, with only quite small numerical differences as well.

Tanaka's experiment uses a 2×2 design testing whether in contexts verifying only an inverse scope reading, QR is more acceptable from embedded subjects vs. embedded objects, as well as whether it is more acceptable from subjunctive vs. indicative clauses, based on a variety of claims made in the theoretical literature. She additionally includes two types of control sentences. The first are monoclausal cases akin to (8a), where acceptability rates are expected to be relatively high. The second are cases involving veritable scope islands like scope-freezing environments and movement islands like the complex NP in (10); acceptability rates for these are expected to be low.

(10) a. * What$_1$ did a technician hear the rumor that Mary inspected t_1?
b. A technician heard the rumor that Mary inspected every plane.

Turning to the results of her experiment, Tanaka finds a significant difference between subjects of subjunctive clauses and objects of indicative clauses, but otherwise finds no statistically significant differences. But interestingly, significant differences were found between the test cases and both types of controls: in all cases, QR from an embedded finite clause was found to be significantly *less* acceptable than QR within the same clause, and significantly *more* acceptable than QR from scope islands. As Tanaka notes, this latter finding is unexpected if finite clauses are true scope islands in the same way that scope-freezing environments and *wh-* islands are; otherwise, we would expect no difference in acceptability. Instead, it seems that QR from finite embedded clauses is not entirely banned, but merely degraded, albeit significantly. There are two *prima facie* plausible ways of accounting for this. First, one could adopt a view of graded grammaticality, with QR from finite embedded clauses being of a low degree of grammaticality, but higher than QR from true scope islands. Alternatively, one could adopt the view that QR from finite embedded clauses is fully grammatical, but difficult to process, leading to lower acceptability rates. We will follow Wurmbrand in adopting the latter view, especially in light of the results on antecedent-contained deletion to which we now turn.

2.1.2 Antecedent-contained deletion

The second type of evidence favoring the existence of QR from finite and non-finite embedded clauses comes from cases of antecedent-contained deletion (ACD). On a traditional account of ellipsis, in order for the elided VP in (11a) to be recovered, it must be copied from the matrix VP, its antecedent (Hankamer & Sag 1976).[7] However, doing this while the object DP remains *in situ* would result in an infinite regress like (11b), since the elided VP is itself contained within the matrix VP. But if the entire object DP undergoes QR outside of the verb phrase as in (11c), then the verb and remaining trace can be harmlessly copied into the elided VP as in (11d) (cf. Sag 1976, May 1985). Such an LF leads to the correct reading that every book that Jesse read, Sam read too.

(11) a. Sam read every book that Jesse did.
b. Sam read every book that Jesse [$_{\text{VP}}$ read every book that Jesse [$_{\text{VP}}$ read every…]]
c. Sam [every$_1$ book that Jesse Δ] $\underline{\text{read } t_1}$.

[7]There are at least two ways of formulating this condition. First, it may be that in the course of the derivation, the elided VP starts off as in some sense null, and at LF this null VP is replaced with a copy of its antecedent ("LF copying"). Alternatively, it may be that the elided and antecedent VPs start as identical during the derivation, with the former being deleted under identity ("PF deletion"). While we frame things in the first way in the body of the paper, the arguments pertaining to ACD and the non-clause-boundedness of QR in no way hinge on this assumption.

d. Sam [every$_1$ book that Jesse read t_1] read t_1.

But there is clear evidence from the theoretical and experimental literature that the QR required for ACD is not clause-bounded, in spite of previous claims to the contrary. For instance, starting again with the theoretical literature, Larson & May (1990) and Kennedy (1997) provide examples where ACD-QR escapes from non-finite embedded clauses, both restructuring and non-restructuring. (12) is an example with the restructuring verb *try*:

(12) Kennedy 1997, p. 673 (his (37)):
Marena usually tries to get a paper accepted at most of the conferences Ted does.

A clearly available reading of (12) is the one that can be paraphrased as in (13):[8]

(13) Marena usually tries to get a paper accepted at most of the conferences that Ted tries to get a paper accepted at.

This reading can only be made available through the embedded object DP QRing above *try*; otherwise, the elided VP will still be contained within its antecedent matrix VP. That is, we need to end up with a structure like (14a), which after copying the antecedent comes out to (14b).

(14) a. Marena usually [most$_1$ of the conferences Ted Δ] tries to get a paper accepted at t_1.
b. Marena usually [most$_1$ of the conferences Ted tries to get a paper accepted at t_1] tries to get a paper accepted at t_1.

This gets the correct reading: most of the conferences Ted applies to, Marena (usually) applies to.

Turning next to non-restructuring verbs, Kennedy provides, among others, the example in (15), with non-restructuring infinitival *ask*:

(15) Kennedy 1997, p. 674 (his (40)):
The First Lady was asked to describe the same documents the President was.

As Kennedy notes, (15) forces a matrix clause interpretation of the ellipsis by means of the clever use of auxiliary choice: an embedded interpretation of the ellipsis (where the antecedent is the VP headed by *describe*) would require *do*-support with *did*, instead of the auxiliary *was*. As a result, the only paraphrase for (15) is the one in (16):

(16) The First Lady was asked to describe the same documents the President was asked to describe.

Much like with (12), this requires the embedded object DP to QR past *ask* so that the elided VP is outside of its antecedent. Thus, ACD-QR must escape the complement of *ask*.

With respect to ACD-QR out of finite clauses, the general consensus in the theoretical literature seems to be that it is at best harder to get than with non-finite clauses, and some (e.g., Kennedy) maintain that it is not possible at all. Nonetheless, Wilder (1997) provides the example in (17), which uses the same mismatched auxiliary technique used by Kennedy in (15):

(17) Wilder 1997, p. 435:
John said that you were on every committee that Bill did.

[8] Wilder (1997) notes that the matrix clause interpretation requires that the object DP be interpreted *de re*, as would be expected on a QR approach to ACD (since the DP outscopes the intensional verb). The reader should bear this fact in mind when looking at (13) and other paraphrases of matrix clause ACD interpretations.

The *do*-support in the elided VP precludes a low-scope reading, since VP-elided copular clauses do not trigger *do*-support. This brings out the matrix reading, paraphrasable as (18):

(18) John said that you were on every committee that Bill said you were on.

As before, this reading is derived by means of the embedded object DP moving past the matrix verb *say*, thereby enabling copying of the matrix VP into the ellipsis site. This therefore entails that the object DP must be able to QR out of a finite embedded clause.

Moving on to the experimental literature, the only work that we are aware of that has directly addressed the issue of adult processing of extraclausal ACD-QR from the complements of embedding verbs is that of Syrett & Lidz (2011) and Syrett (2015a,b).[9] Syrett & Lidz (2011) perform two relevant experiments in this area, each of which tested both adults and children. However, for this and the other paper discussing experiments with both adults and children (Syrett 2015b), we will report only the results from adults, as we restrict our focus in this paper to adult parsing of extraclausal QR. In the first of the two relevant experiments, Syrett & Lidz test for the possibility of ACD-QR from non-finite embedded clauses. They use sentences that are (hypothetically) ambiguous—that is, not disambiguated via auxiliaries like Kennedy and Wilder's examples above—such as (19):

(19) Syrett & Lidz 2011, p. 315 (their (24b)):
Clifford asked Goofy to read every book that Scooby did.

The *matrix* reading of (19) results from long QR and copying of the matrix VP headed by *ask*, while the *embedded* reading results from copying only the embedded VP (headed by *read*) into the ellipsis site. Syrett & Lidz place sentences like (19) in contexts in which only one of these two potential readings is true, with participants asked to indicate whether the sentence is true in the context, as well as to provide justification for their answer.

In their second experiment, Syrett & Lidz perform essentially the same task, but replacing non-finite embedded clauses with finite ones, as in (20):

(20) Syrett & Lidz 2011, p. 321 (their (26)):
Clifford said that Goofy read every book that Scooby did.

Once again, participants were asked to provide truth value judgments and justifications for their answers, in contexts that verified either the matrix or embedded reading (but not both).

The results of Syrett & Lidz's first experiment show that adults were clearly able to obtain both embedded and matrix readings for ACD in non-finite clauses, but that matrix readings were harder to obtain than embedded readings: adults responded "true" 68% of the time in contexts where only the embedded reading was true, and 50% of the time in contexts where only the matrix reading was true. These findings are further bolstered by participants' justifications for their responses, which reliably pointed to their answers being a reflection of their accessing a matrix or embedded reading.

[9] Hackl et al. (2012) also study ACD-QR out of non-finite clauses, but these clauses are the complements of adjectives rather than verbs, as in the following example:

(i) Hackl et al. 2012, p. 173 (from their (29)):
The doctor was reluctant to treat every patient that the recently hired nurse was.

We do not tackle clausal complements of adjectives in this paper, but it is worth noting that their findings fit well with the general picture arrived at in this section: unambiguous wide-scope readings like the above example are harder to process than unambiguous low-scope readings of the sort obtained by replacing *was* with *did*.

When turning to finite embedded clauses in the second experiment, designation of truth fell to 19% in contexts verifying only the matrix reading, in contrast to 88% in contexts verifying only the embedded reading. Thus, while some participants were able to access the matrix reading in finite embedded clauses (with varying degrees of consistency), the second experiment suggests that at best, adults found matrix readings for ACD in finite embedded clauses considerably harder to get than in non-finite clauses.[10]

Acknowledging that Syrett & Lidz's (2011) evidence for matrix readings of finite-clause-embedded ACD is relatively weak, Syrett (2015b) sets out to provide more definitive evidence that this reading is available for both children and adults. She does so by making a variety of tweaks meant to make obtaining a matrix reading easier for the participants. In addition to subtly revising the context, this included several grammatical changes, such as removing the complementizer *that*—the matrix verb was again *say*, with which *that* is optional—and using prosodic focus in a way that may be more conducive to bringing out a matrix reading. Like in Syrett & Lidz's (2011) study, these sentences avoided any grammatical manipulations that would fully disambiguate them, and participants were asked to provide truth value judgments and justifications for their answers, in contexts that exclusively made either the embedded or matrix reading true. Moreover, the adult participants' answers were filtered in a way that ensured that the tallied results reflected actual interpretations: answers were only counted if they were paired with justifications that clearly pointed to their accessing the reading coinciding with their answer.

The adults' answers provided much more powerful evidence that they were indeed able to access matrix readings with finite embedded clauses, with 74.2% providing justified "true" answers in contexts where only the matrix reading was true. Moreover, participants showed a preference for matrix readings even in contexts where only the embedded reading was true: 64.9% provided justified "false" answers in such contexts. This strongly supports the claim that DPs can QR from within a finite clause to a position above the matrix verb. In fact, Syrett (2015a) ups the ante, using a similar task to provide evidence that not only can the DP QR to a position above the matrix verb, but it can QR even higher to a position that outscopes the subject of the matrix clause. That is, (21) allows a reading where different speakers can make claims about different frogs:

(21) Syrett 2015a, p. 585 (her (14)):
Someone$_i$ said he$_i$ could jump over every frog that Jessie did.

However, this "extrawide" scope seems to come at a cost over and above that required to get a matrix reading, as less than half of the responses indicated an inverse scope interpretation.

2.1.3 Wurmbrand's conclusions

Given the above evidence, Wurmbrand reaches the following empirical conclusions. First, she concludes that as far as grammatical competence is concerned, QR is an unbounded operation, and a quantificational DP can QR as high as it wants so long as it obeys the normal rules of A′-movement. Cases where DPs seem incapable of scoping out of embedded clauses really amount to situations where computing the relevant (well-formed) syntactic representation is too costly for the human parser to process.

[10]For critical discussion of Syrett & Lidz's experiments, see Sugawara et al. 2013. However, the latter's criticism focuses mostly on the results from children, whose justifications for their responses were considerably less reliable than those of the adults.

Second, Wurmbrand concludes that while a variety of factors can ease the processing of inverse scope, movement that impacts LF is nonetheless costly to process, as evidenced by the fact that surface scope is in general easier to process than inverse scope. (For embedded DPs the higher cost of inverse scope is made clear from the experimental studies cited above; in the case of within-clause inverse scope, see Kurtzman & MacDonald 1993, Tunstall 1998, Anderson 2004.)

Third, Wurmbrand concludes—appropriately tentatively—that inverse scope is easier from within the same clause than it is across clausal boundaries, and that in the case of non-finite clausal complements QR from the complements of *try*-type verbs is easier than from the complements of *decide*-type verbs. That is, inverse scope is easier for (8a) than for (8b), which is easier than for (8c).

(8) a. A technician inspected every plane.
 b. A technician tried to inspect every plane.
 c. A technician decided to inspect every plane.

QR from finite clauses, much like their non-finite counterparts, is significantly more difficult than for monoclausal cases. However, Wurmbrand avoids a concrete analysis of the relative difficulty of QR from finite clauses because there is little if any experimental work directly comparing QR from finite and non-finite embedded clauses. Moreover, there are additional factors that may come into play with finite complements, such as mood (cf. Tanaka 2015b for discussion), or the presence or absence of an overt complementizer (Syrett 2015a,b). While the discussion from the theoretical literature seems to at least suggest that QR from finite clauses is harder than QR from any kind of non-finite clause, we will nonetheless follow Wurmbrand in being non-committal on this front and sticking to the cases in (8), in addition to further data to be introduced shortly.

2.2 Wurmbrand's analysis

2.2.1 The different sizes of non-finite clausal complements

At the heart of Wurmbrand's explanation for the relative processing difficulty of inverse scope in the sentences in (8) is a theory of non-finite clausal complementation, developed most notably in Wurmbrand's prior work (see, e.g., Wurmbrand 2001, 2014a,b, 2015), in which different verbs have different possibilities for the size of complement clause that they take. More specifically, on this view verbs taking infinitival complements are thought to fall into three classes, which can be differentiated by their syntactic and semantic properties.

The verbs that take the largest non-finite complements are verbs like *believe* and *claim*, whose complements she argues are full proposition-denoting CPs, which include all the heads along the clausal spine that one normally sees below C: namely, Tense/Aspect/Modality (TAM) and of course the θ-assigning heads *v* and V:

(22) Becca claimed [$_{\text{CP}}$ to be in Boston].

The next largest non-finite complements are those of "future-shifting" verbs like *decide* and *expect*. The complements to these verbs lack a CP layer, but include at least one TAM head—the future-shifting WOLL—along with *v* and V. Out of a desire to remain neutral on what the label of these complements is, we will simply refer to it as WOLLP:

(23) Becca decided [$_{\text{WOLLP}}$ to go to Boston].

Finally, the verbs that take the smallest complements are *try*-type verbs, whose complements lack both a CP layer and WOLLP. Thus, for these verbs, the complement is only a *v*P:[11]

(24) Becca tried [$_{v\text{P}}$ to go to Boston].

A diagram of the heads of the clausal spine posited to be present in the non-finite complements of these three types of verbs can be seen in (25):

(25)
$$\underbrace{\text{C}\qquad\underbrace{\text{TAM}\qquad\underbrace{v\qquad\text{V}}_{\textit{try}\text{-type}}}_{\textit{decide}\text{-type}}}_{\textit{claim}\text{-type}}$$

In prior work, Wurmbrand has argued for such variation in the sizes of non-finite complements on both syntactic and semantic grounds. We will not go through her arguments, but just as a sampler, the presence of WOLL with *decide*-type verbs but not *try*-type verbs helps explain differences in the presence/absence of future-shifting in the complement, as evidenced by examples like (26):

(26) Yesterday, Becca {decided/#tried} to go to Boston tomorrow.

Moreover, the subset/superset relation with respect to heads along the embedded clausal spine is used to account for an important implicational hierarchy. As discussed in detail by Wurmbrand (2014a, 2015), extraclausal scrambling and clitic climbing—two generally accepted diagnostics for restructuring environments—vary in their distribution crosslinguistically. More specifically, while some languages disallow these operations entirely, some allow these operations only from *try*-type infinitives, and some allow them from both *decide*- and *try*-type infinitives, there appear to be no languages that allow them from only *decide*-type infinitives. There also appear to be no languages that allow these operations from *claim*-type complements, nor from finite clausal complements.[12]

As Wurmbrand shows, this implicational hierarchy gets a ready account if non-finite complements look as in (22–24). Say that there is some structural configuration Σ that blocks both clitic climbing and scrambling, and say that in Language A, Σ resides inside the *v*P. Since the complements of all three types of verbs contain at least a *v*P, in this language Σ will appear across the board, meaning that clitic climbing and extraclausal scrambling will be non-existent. Meanwhile, if in Language B Σ appears in the TAM field above *v*P, it will be present in the complements of *decide*- and *claim*-type verbs, but not *try*-type verbs, meaning that the latter (and only the latter) will be transparent to clitic climbing and scrambling. Finally, if in Language C Σ is above TAM in the CP field, it will only be present in the complements of *claim*-type verbs, meaning that both *decide*- and *try*-type verbs will be transparent to clitic climbing and scrambling. The implicational hierarchy falls out immediately, since there is no place that Σ can be located that would block clitic climbing and scrambling from *try*-type complements without also blocking them from *decide*-type complements.

[11] Wurmbrand leaves open the possibility that lower aspectual heads may exist between WOLL and *v*, and that these may be present or absent with *try*-type infinitives. This is not important for her analysis, nor for ours.

[12] Wurmbrand (2014a) notes that cross-linguistically, the long passive (cf. (9)) seems to be available only with *try*-type complements, and its (im)permissibility in a given language seems to be orthogonal to the possibility and maximal length of clitic-climbing and extraclausal scrambling. For this reason she argues that the long passive is an altogether different type of restructuring, with an altogether different syntactic origin, but one whose basis still lies in varying sizes of clausal complements.

2.2.2 *Processing difficulty and movement-counting*

Wurmbrand accounts for the difference in inverse scope processing difficulty for the sentences in (8) by adopting the view of non-finite clausal complementation detailed above, and simultaneously positing that movement that affects the LF structure is costly for the human parser to process. More specifically, she posits that each scopally relevant movement incurs a particular cost, with the difficulty in scope processing for a given parse being proportional to the number of such movements that take place.

To see how this—in conjunction with certain constraints on A′-movement—generates the right results with respect to the contrast between the sentences in (8), let us first consider the monoclausal case (8a) (*A technician inspected every plane*). Assuming that the subject is merged in spec-*v*P, the least LF-movement-heavy way to generate inverse scope would be for the subject to scope in its merge position, with the object taking scope just above it at the edge of *v*P. This is illustrated in (27); heads above the (matrix) *v*P are excluded for readability:

(27) $[_{vP}$ [every plane]$_1$ λx_1 a technician inspect $\boldsymbol{t}_1]$

As indicated by the single emphasized trace, inverse scope for (8a) requires one LF-impacting movement.

As for (8b) (*A technician tried to inspect every plane*), recall that for Wurmbrand the complement of *try* is a *v*P. It is commonly thought that *v*P constitutes a domain for movement: in order for a constituent to move out of *v*P, it must first move to the edge of *v*P, and then move from there to its destination (or to the edge of the next-highest domain). Thus, as seen in (28), in order for *every plane* to scope above *a technician*, it must undergo QR twice: first to the edge of the embedded *v*P, then to the edge of the matrix *v*P:

(28) $[_{vP}$ [every plane]$_1$ λx_1 a technician try $[_{vP}$ $\boldsymbol{t}_1$ v inspect $\boldsymbol{t}_1]]$

Finally, there is (8c) (*A technician decided to inspect every plane*). To account for this case, Wurmbrand proposes that in the same way that *v*P is a domain for movement, WOLLP is also a domain for movement, at least in the syntactic configuration seen in (8c).[13] Thus, in order for *every plane* to outscope *a technician*, it must QR three times: first to the edge of the embedded *v*P, then to the edge of the embedded WOLLP, and then to the edge of the matrix *v*P, as in (29):

(29) $[_{vP}$ [every plane]$_1$ λx_1 a technician decide $[_{\text{WOLLP}}$ $\boldsymbol{t}_1$ WOLL $[_{vP}$ $\boldsymbol{t}_1$ v inspect $\boldsymbol{t}_1]]]$

Therefore, given Wurmbrand's theory of non-finite clause sizes, in conjunction with reasonable assumptions about the nature of A′-movement from embedded clauses, her principle of scope processing difficulty generates the right predictions with respect to the sentences in (8), i.e., inverse scope for (8a) is easier than for (8b), which is easier than for (8c).

Before discussing problems with Wurmbrand's theory of scope processing difficulty, one advantage to her theory is worth emphasizing: its strong predictive power. A great deal of work in quantifier scope processing has focused on a fairly narrow question: *Given a scopally ambiguous sentence S, what determines the preferred or default reading for S?* Take, for example, Tunstall's (1998) Principle of Scope Interpretation (PSI):

[13] Wurmbrand does not provide direct empirical evidence for this claim, but for our purposes this is irrelevant: as discussed in §5.4, our own analysis is compatible with, but does not require the assumption that WOLLP is a movement domain.

> “The default relative scoping in a multiply quantified sentence is computed from the required LF-structure of that sentence, where the required LF is determined by the required grammatical operations acting on the S-structure. The default scoping is the preferred scoping unless there is evidence to go beyond it.” (Tunstall 1998, p. 56)

The PSI offers an explanation for why each of the sentences in (8) is easier to process on a surface scope reading than on an inverse scope reading. However, by virtue of the fact that it focuses on comparing different readings for the same sentence, the PSI cannot explain why inverse scope is harder for some sentences than for others. Meanwhile, Wurmbrand’s analysis allows for precise comparative predictions not only for different readings of the same sentence, but also for different readings of *different* sentences, such as those in (8); this is an admirable trait that ought to be preserved in any explanatory theory of scope processing.

However, in spite of these positives, we will next argue that Wurmbrand’s theory of scope processing difficulty runs into some trouble when looking beyond the particular cases she analyzes.

2.3 The problematic impact of overt movement

On Wurmbrand’s theory, processing cost is calculated by counting the number of movements that affect LF structure: there is essentially a “tax” on LF traces, so that each such trace imposes a processing cost on the human parser.[14] Thus, the best evidence against Wurmbrand’s analysis would come from cases where LF-relevant movement does not appear to be a particularly costly operation, or even better, cases where a parse involving more LF traces is *easier* than a parse involving fewer.

One such apparent counterexample is noted by Wurmbrand herself: her analysis as it currently stands has trouble when it comes to *wh*-movement. Consider, for example, the sentences in (30):

(30) Wurmbrand 2018, p. 25 (based on her (29)):
  a. What did a technician say that John inspected?
  b. A technician said that John inspected every plane.

In (30a), *what* moves cyclically from the complement of *inspect* to some specifier in the left periphery of the matrix clause; let us say it is the specifier of CP. Based on the received view that *wh-* phrases that overtly move to the left periphery also take scope there (Karttunen 1977 and much work since), we can infer that each of these movement operations also leaves a trace at LF. Meanwhile, as discussed above, in order for *every plane* to scope above *a technician* in (30b), it only has to QR to the edge of the matrix *v*P. Thus, a well-formed interpretation for (30a) requires at least as many iterations of scope-taking movement as an inverse-scope interpretation of (30b)—possibly more, since *what* has to move from the edge of the matrix *v*P to spec-CP, a move we assume is not required for the inverse scope of (30b). Thus, Wurmbrand predicts (30a) to be at least as difficult to process, if not more so, than an inverse scope interpretation of (30b). But while we are not aware of any experimental studies directly addressing this question, at least on an intuitive level things appear to be quite the opposite: (30a) is noticeably *easier* to process than the inverse scope of (30b). Of course, relying on intuitions to make judgments of processing difficulty is questionable

[14]In fact, Wurmbrand’s analysis could more accurately be referred to as “trace”-counting rather than LF-movement-counting. She assumes a copy theory of movement (Chomsky 1995), using Fox’s (2002, 2003) *trace conversion* operation to convert lower copies into what can be semantically interpreted like traces. Processing cost is linked to the number of iterations of trace conversion, i.e., to the number of “traces”. For arguments against trace conversion and a compositional semantics within the copy theory of movement that obviates it, see Pasternak 2019.

to say the least, but pending much-needed experimental investigation comparing and contrasting such cases, we maintain that a model that predicts (30a) to be easier than the inverse scope of (30b) is *prima facie* more likely to be correct than one that predicts the opposite.

Another issue faced by Wurmbrand's account—and one with much stronger experimental evidence to back it up—pertains to the relative scope of sentential negation and universal quantifiers, as explored in detail by Lee (2009). Lee tests such scope preferences in both English and Korean, for universal quantifiers in both subject and object positions.[15] Some relevant examples in both languages can be seen in (31) and (32):[16]

(31) ***Every* in subject position**

a. According to the story, every kid didn't feed the doves in the park. (Lee 2009, p. 93)

b. hwancangsil-eyse motun haksayng-i son-ul an ssis-ess-ta-ko iyaki-un
in the restroom every student-NOM hand-ACC NEG wash-PST-DECL-COMP story-TOP
malhaycwunta
tell
'The story tells that every student did not wash her hands in the restroom.'
(Korean, Lee 2009, p. 79)

(32) ***Every* in object position**

a. According to the story, Cindy didn't light every candle last night. (Lee 2009, p. 124)

b. eey pam-ey Sehee-ka motun chospwul-ul an khye-(e)ss-ta-ko iyaki-nun
last night Sehee-NOM every candle-ACC NEG light-PST-DECL-COMP story-TOP
malhaycwuntu
tell
'The story tells that Sehee did not light every candle last night.'
(Korean, Lee 2009, p. 112)

English and Korean provide an intriguing point of comparison and contrast due to their distinct word orders: while subjects in both languages linearly precede negation, direct objects occupy different positions in the two languages, with Korean objects preceding negation and English objects following it. These similarities and differences are reflected in the results of Lee's experiments. Lee finds that for both English and Korean speakers, there is a preference for subject *every*-DPs to scope over negation, as indicated by both truth value judgments and reading times. Meanwhile, in the case of direct objects, Korean and English diverge: Korean speakers preferred to parse *every* over negation, while English speakers were more inclined to parse such sentences with a *not* > *every* reading.[17] In other words, scope preferences more or less reflect linear word order: English

[15]Lee also tests native Korean-speaking L2 speakers of English. While these results are fascinating, we will not account for them in this paper, as it is not sufficiently clear what a Korean L2 grammar (let alone parser) of English looks like.

[16]As Lee discusses in detail, Korean has two forms of sentential negation: "long form" and "short form". The examples in (31b) and (32b) utilize the short form; Lee tests both forms, and finds similar results for each.

[17]One might reasonably be concerned about the results from English direct objects, since in such sentences the "favored" reading (*not* > *every*) is weaker than the "disfavored" one (*every* > *not*). However, Lee notes that scalar implicatures eliminate this entailment relation. For example, the surface scope reading of (32a) generates an implicature that Cindy lit at least some candles; not only is this implicature not entailed by the inverse scope reading, but it contradicts it. Lee's test items exploit these implicatures in a way that avoids concerns about inappropriate entailments.

subjects and Korean subjects and objects precede negation and prefer to scope over it, while English objects follow negation and prefer to scope beneath it.

Wurmbrand's analysis could plausibly explain the case of English direct objects, since the preferred reading is the one in which the direct object scopes closer to its merge position. However, it fails to account for the behavior of English subject DPs, and at least on certain assumptions, Korean subjects and direct objects as well. To see why, consider the case of English subjects. It is generally thought that in English clauses with sentential negation, the subject is initially merged below negation and subsequently undergoes overt movement past negation to spec-TP:

(33) [$_{\text{TP}}$ [every kid]$_1$ not [$_{v\text{P}}$ t_1 feed the doves in the park]]

The *every* > *not* reading is the result of *every kid* scoping in this position above negation, while the *not* > *every* reading is likely the result of *every kid* reconstructing to its merge position.

So which of these does Wurmbrand predict to be easier? In the case of the surface scope interpretation, there is one LF-relevant movement (spec-*v*P to spec-TP). As for the inverse scope interpretation, it depends on how one thinks reconstruction works. In the days when reconstruction was achieved through an LF movement operation of Quantifier Lowering (May 1985), the *not* > *every* interpretation would be generated via two movements: the (overt) movement above negation, and the (covert) movement back down. In this case, Wurmbrand's analysis would make the right prediction, since surface scope would require one LF-affecting movement, while inverse scope would require two. But the more common view nowadays is that *every kid* does not reconstruct below *not* by undergoing a downward LF movement operation, but instead does so by just not moving above it in the first place: by some means or another, the movement of *every kid* above negation impacts PF, but not LF. Given this analysis, the *not* > *every* reading is generated with zero LF-relevant movement steps, meaning that Wurmbrand incorrectly predicts that English subjects should prefer to scope under, rather than over negation.

Turning to the Korean facts, things are somewhat more complicated, as it depends on how one wishes to generate the S-O-(Neg)-V word order. However, a plausible view—especially if one adheres to an antisymmetric theory of syntax, in which linear order is determined by c-command (Kayne 1994)—is that the subject and object start below (and, linearly, to the right of) negation, much like in English, and then both undergo overt movement past negation. This could be done either via direct movement of both DPs to positions c-commanding negation, or via remnant movement, or some combination of the two. Either way, on the assumption that these DPs can scope in their surface position above negation, and would scope under negation via reconstruction, Wurmbrand's theory of scope processing difficulty fails to account for the processing facts for the same reason as in the case of English subjects: reconstruction entails fewer LF traces than scoping at the post-movement position, but nonetheless the latter is preferred.

While these two problematic cases—*wh*-movement and negation—look quite different, both illustrate that scope processing difficulty cannot be determined based solely on LF structure. Instead, whether a movement that affects LF also affects PF seems to have a crucial impact on processing difficulty. In the case of *wh*-movement, the fact that *wh*- phrases move overtly significantly eases processing. The cases with quantificational DPs and sentential negation paint an even more extreme picture: overt scope-taking movement is not only easier to process than movement that affects only LF (QR), but is also easier than movement that affects only PF (reconstruction). Thus, when English subject DPs move past negation and also take scope above negation, the result is easier to process than when they move past negation only at PF and stay below negation at LF.

As a first gesture towards accounting for a potential difference between *wh*-movement in (30a) and QR in (30b), Wurmbrand observes the following:

> Overt *wh*-movement involves a filler-gap dependency where upon encountering the filler (the overt *wh*-phrase), the parser is instructed to look for a gap (the originating position). In doubly quantified sentences, on the other hand, there is no overt cue within the sentence for a long-distance dependency until the second QP [quantifier phrase] is reached. Thus, in contrast to overt *wh*-movement, QR involves a retrospective search in parsing, which could be seen to be responsible for the higher processing cost for covert movement. (Wurmbrand 2018, p. 25)

But there are two problems with this proposal. The first is that in contrast to Wurmbrand's original account in terms of LF-traces, this processing-based proposal is not sufficiently worked out to make strong, quantitatively grounded predictions. The second is that it is unclear how this analysis could handle the case of English subjects scoping over negation. After all, when the parser runs into a DP subject, it should know that there is a gap at the subject's merge position (just like in the *wh*- case), and that the subject could take scope either where it is pronounced or at that merge position. There should thus be no retrospective search problem, and *not* > *every* should be at least as accessible as *every* > *not* when overt movement has taken place, contrary to fact.

More promising is an idea that Wurmbrand suggests in passing in a couple of footnotes (her fns. 20 and 25): namely, that there is an additional processing cost when there is a mismatch between PF and LF. In fact, if we allow ourselves to talk about PF-LF mismatches in scalar terms—that is, in terms of greater or lesser mismatches between PF and LF—then this principle would on its own be enough to account for the facts discussed. In the case of the sentences in (8), we rightly predict a preference for surface scope over inverse scope for each sentence, since surface scope is the interpretation in which LF most closely aligns with PF. Moreover, if we define severity of mismatches correctly, we can point to why inverse scope for (8a) is easier than for (8b), which is easier than for (8c): in later examples, the object DP has to scope farther and farther away from where it is pronounced, creating more severe PF-LF mismatches, and thus imposing greater processing costs. These costs are not incurred in the case of *wh*-movement, since the *wh*- phrase scopes at its left-periphery PF position. In the case of DPs and sentential negation, when a DP overtly moves past negation, we expect an *every* > *not* reading to be easiest, and when the DP stays below negation at PF, we expect a *not* > *every* reading to be easiest, expectations that accord with Lee's (2009) findings from English and Korean.

This, in a nutshell, is the approach we will adopt in the rest of this paper. Such an approach carries with it two concomitant questions. First, how do we formally define a scalar notion of PF-LF mismatch with a sufficient degree of predictive power? And second, what is it about the nature of the human parser that entails that this particular property of syntactic representations (or derivations) should lead to processing difficulty? In order to bring us closer to an explanatory theory of scope processing difficulty, we will approach these questions from a formal parsing perspective: we will provide a formal grammar that generates both PF and LF representations, along with a formal parser for that grammar, and we will show that scope processing difficulty correlates with a particular type of information storage arising during the course of a successful parse. The formal grammar utilized will be a version of Stabler's (1997) Minimalist Grammars (MGs); as discussed in the introduction, MG parsing has been used to account for a variety of syntactic processing phenomena, and our theory serves as a significant extension of this already productive research program. §3 will introduce MGs, starting with a simpler PF-only version, then moving on to a version that also generates LF structures. §4 will introduce a simplified version of our LF-inclusive MG parser; a completely formally fleshed out version can be found in the appendices.

3 A semi-formal introduction to Minimalist Grammars

In this section we will introduce Minimalist Grammars (MGs, Stabler 1997), starting with a simpler PF-only variant, then moving on to a version that simultaneously generates both PF and LF outputs. In §3.1 we will introduce the basics of the PF-only MG, and in particular its use of feature-driven merge and move operations to build and manipulate syntactic structures. In §3.2 we will discuss and illustrate an alternate way of performing these same operations in terms of *chains*. In short, on a traditional approach to movement, a constituent first merges in some position, then moves to another position and leaves a trace in its original spot. On a chain-based view of movement, the first step is skipped: the trace of a "moved" constituent is simply merged in its place from the get-go, with the moved constituent being held onto until it reaches its final landing spot, at which point it is finally merged into the tree. The advantage of a chain-based approach for our purposes is that the chains involved in a derivation will bear a strong resemblance to—and have a one-to-one correspondence with—the parse items involved in the corresponding parse. In §3.3 we will add LF representations into the mix: MG derivations will simultaneously build a PF tree and an LF tree along the lines of Heim & Kratzer 1998. We will again illustrate by means of an example: namely, a derivation of the inverse-scope interpretation of (8a) (*A technician inspected every plane*). Finally, in §3.4 we will introduce *derivation trees*, tree structures representing the course of a syntactic derivation; these will be useful in representing both derivations and the parses of those derivations.

In discussing MGs, and especially in going over MG parsers in the next section, we will try to keep things as informal as possible: the goal in the body of the paper is only to give enough details for the reader to understand the scope processing metric in §5 and the predictions it makes. A completely fleshed-out formalism is left to the appendices.

3.1 PF-only MG: features and operations

In accordance with the Minimalist Program, MGs are highly lexicalized formal grammars: lexical items come bundled with strings of features that must be checked over the course of a derivation. More specifically, lexical items can be treated as ordered pairs (A, δ), where A is a phonetic form (for which we will simply use English orthography) and δ is a string of features. We will sometimes write this as $\mathrm{A} :: \delta$. Features come in four sorts:

- *Category* features (e.g., `f`) can roughly be thought of as indicating the type of phrase headed by the lexical item (e.g., `D` for DP, `T` for TP, etc.).
- *Selector* features (`=f` or `f=`) indicate the type of phrase taken as a complement or specifier. `=f` indicates that an f-phrase merges to the right (i.e., as complement); `f=` indicates that it merges to the left (specifier).[18]

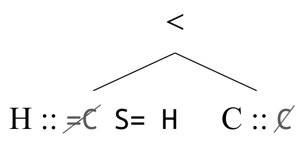
[18]Standard MGs dating back to Stabler 1997 only include one type of selector feature, with linearization being determined by order of merge: the first selected-for phrase merges to the right (complement), and subsequent phrases merge to the left (specifier). However, encoding left-selection and right-selection through separate selector features does not affect generative capacity and leads to a simpler parser, so we adopt it for the sake of convenience.

- *Movement licensor* features (`+f`) license movement of lower phrases to a specifier position.
- Corresponding *movement licensee features* (`-f`) are included with the heads of moving phrases.

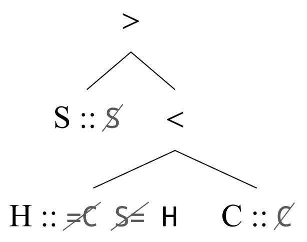
A feature is checked via deletion: if a head has a feature string $\phi_1\phi_2\phi_3$, then after checking ϕ_1 the head will have the feature string $\phi_2\phi_3$. Features are always checked from left to right; we will refer to a given head's leftmost feature as its *active feature*.

MGs allow for two types of operations, the execution of which is dependent on the active features of relevant lexical items. The first is **merge**, in which a phrase whose head's active feature is a selector feature combines with a phrase whose head's active feature is a matching category feature. As indicated above, whether the selected-for phrase merges to the right (complement) or left (specifier) depends on the choice of selector feature.

As a toy example, suppose we have the lexical items H :: `=C S= H`, C :: `C`, and S :: `S`. H's active (i.e., leftmost) feature is `=C`, while C's active feature is `C`. Thus, the two can merge into the tree seen in (34), where the non-terminal node < indicates that the head of the constituent is to the left. Notice that since the merge operation checks the selector and category features, each feature is deleted in the resulting tree. (Just for these initial examples, deleted features will be crossed out; in future examples they will simply be removed from the feature string.)

(34)


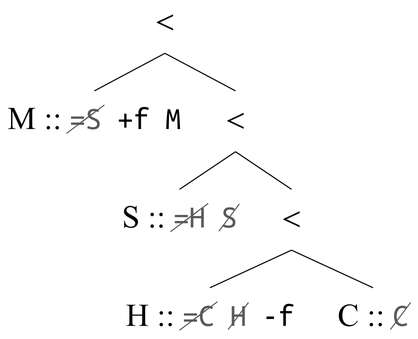
Now C has no active features, and H's active feature is the selector feature `S=`, which matches S's category feature `S`. Since the selector feature is `S=` and not `=S`, S will merge to the left of the tree built so far, resulting in (35):

(35)


The only remaining active feature is H's category feature `H`. Since H is the head of this phrase, this phrase can be selected as a complement or specifier of some other head with the feature `=H` or `H=`.

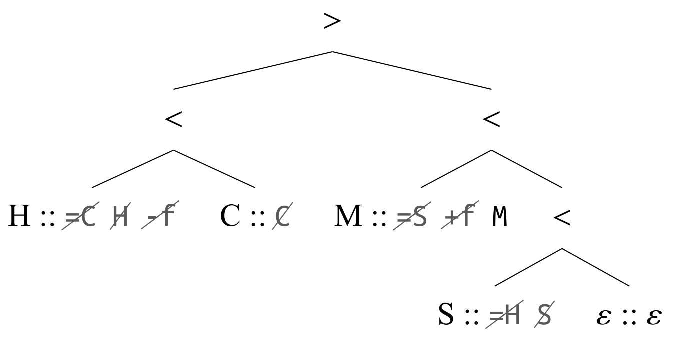
The second operation in MGs is **move**, which takes place when the head of the currently derived phrase has as its active feature a movement licensor feature `+f`, and the head of a subphrase has as its active feature a movement licensee feature `-f`. In this case, the maximal subphrase having the latter lexical item as its head moves up to a newly-created specifier position, leaving behind a phonologically empty and featureless node. Take, for example, the tree in (36):

(36)



M, the head of this phrase, has as its active feature the licensor feature `+f`, and H has an active licensee feature `-f`. As a result, the maximal phrase headed by H moves to a newly-created specifier, leaving an empty node behind. (ε is the empty string.) Since the maximal phrase headed by H is the phrase containing H and C (and not S), this looks like (37). Note once again that the licensor and licensee features are deleted, meaning that the only active feature is the category feature `M` of M, the head of the phrase as a whole.

(37)



3.2 Chain notation

Up to this point we have assumed that if one phrase merges with another and then moves, this is precisely what happens: the phrases combine, and then the moving phrase moves up to a specifier and leaves behind an empty node. But there is a bit of redundancy in this. If phrases A and B merge, and B will subsequently be moving, we already know at the time of merging that B will eventually be moving, since its next active feature will be a movement licensee feature. As a result, we could skip the middle man: rather than merging A and B, then eventually moving B and adding the empty node in B's original location, we could instead simply combine A with the empty node to begin with, holding on to B until we reach B's landing site, at which point B would be added. In cases where B moves multiple times, we would continue to hold onto it until the very last movement is done, with empty "traces" being inserted during the course of intermediate movement operations.

In order to accomplish this, we need to add a little more technical machinery, enough to define this notion of "holding on" to a moving subtree. For this purpose we will use a "chain" notation similar to that used by Stabler & Keenan (2003). We define a *feature-specified tree (FST)* as an ordered pair (τ, δ), where τ is a tree, and δ is a string of features; this string of features will be the remaining unchecked features of the head of τ. A *chain* is an ordered tuple $\langle(\tau_1, \delta_1); (\tau_2, \delta_2) \ldots (\tau_n, \delta_n)\rangle$ of FSTs. The first member of this tuple will be the *primary FST*, which is the FST of the main tree that we are building; the other FSTs in the tuple are the ones that are being held onto for later addition into the tree, i.e., the movers. A lexical item is a unary chain, i.e., a chain with just one FST. The tree in this FST will consist of one node that is the phonetic (in our case, orthographic) representation of the word, while the feature string will be the features of this lexical item. (One-node trees will simply be written as the phonetic representation of the single node in that tree.)

To illustrate, let us go through a derivation for the simple sentence *Kat kicked Matt*. Before doing so, one important fact is worth noting: to avoid unnecessary complexity, we will not be dealing with any sort of head movement in this paper. MGs and their parsers are fully capable of handling head movement, and there is little reason to believe that the modest extension to MGs offered in §3.3 should suddenly be incompatible with head movement. However, we believe that there is equally little reason to think that including head movement would have any significant impact on the results in this paper, so to keep complexity to a minimum we exclude it. With this in mind, the enumeration for our simple sentence can be seen in (38):

(38) a. ⟨(Matt, `D`)⟩ c. ⟨(ε, `=V D= v`)⟩ e. ⟨(ε, `=v +nom T`)⟩
b. ⟨(kicked, `=D V`)⟩ d. ⟨(Kat, `D -nom`)⟩ f. ⟨(ε, `=T C`)⟩

As should be clear, (38a), (38b), and (38d) are the lexical items *Matt*, *kicked*, and *Kat*, respectively. (38c) is the lexical entry for *v*, (38e) the entry for T, and (38f) the entry for C. Note that for these three, which are unpronounced, the phonetic representation is the empty string ε.

We can now start building our tree. First, we merge *kicked* and *Matt*. *Matt*'s feature string contains only the category feature `D`—*Matt*, being a proper name, is a one-word DP—and no movement licensee features, meaning that once it is merged it will not undergo any subsequent movement. We thus do not need to “hold on” to it for future movement operations, which means that the unary chains for *kicked* and *Matt* can be merged into another unary chain to form the VP *kicked Matt*, as in (39). Here and throughout, the first argument of **merge** will be the chain with the selector feature, with the second being the chain with the category feature. For readability we rewrite **merge**(A, B) as A **merge** B.

(39) (38b) **merge** (38a) = ⟨([$_<$ kicked Matt], `V`)⟩

Both the selector feature `=D` and the category feature `D` are deleted. The lone FST in the resulting chain has as its only feature `V`, the category feature of the head *kicked*. Since the VP will not undergo subsequent movement after being merged with *v*—it lacks a movement licensee feature—it can be directly merged with *v*, which has the appropriate selector feature `=V`, in the same manner:

(40) (38c) **merge** (39) = ⟨([$_<$ ε [$_<$ kicked Matt]], `D= v`)⟩

The next step is to merge the subject, *Kat*, in the specifier of *v*P. Notice that *Kat*, in addition to its category feature `D`, also has the movement licensee feature `-nom`, meaning that it will undergo movement after merging. Thus, as per the discussion above, rather than merging *Kat* into spec-*v*P and subsequently moving it, we will simply put an empty node in spec-*v*P and add the FST in the lexical entry for *Kat* at the end of our chain, giving our first non-unary chain. The category and selector features `D` and `D=` are again deleted.

(41) (40) **merge** (38d) = ⟨([$_>$ ε [$_<$ ε [$_<$ kicked Matt]]], `v`); (Kat, `-nom`)⟩

We next merge this chain with T, deleting the selector and category features `=v` and `v` and passing the rest of T's features up to the primary FST of the new chain. The second element of the *v*P's chain, *Kat*, is passed along unchanged to the newly-formed chain, still waiting for its time to be tacked onto the primary FST.

(42) (38e) **merge** (41) = ⟨([$_<$ ε [$_>$ ε [$_<$ ε [$_<$ kicked Matt]]]], `+nom T`); (Kat, `-nom`)⟩

Up to this point, the active feature of the primary FST has always been either a selector feature or a category feature. This is why until now we have only performed **merge** operations, and no **move** operations. But now, for the first time the active feature of the primary FST is a movement licensor feature (`+nom`), meaning that movement will take place. *Kat* has the corresponding licensee feature (`-nom`), meaning that it will move to spec-TP. This entails deleting the movement licensor and licensee features—again, features are always checked by deletion—as well as placing *Kat* in spec-TP and removing the second FST from the chain:

(43) **move** (42) = ⟨([$_>$ Kat [$_<$ ε [$_>$ ε [$_<$ ε [$_<$ kicked Matt]]]]], `T`)⟩

Two things are particularly worth noting about this step. The first is that we are back to a unary chain: now that *Kat* has finished moving, we no longer have any movers awaiting their final landing spot. The second is that whereas **merge** was a binary operation, taking two chains and combining them into one, **move** is a unary operation, converting one chain into another. This makes sense given the nature of **merge** and **move** as tree operations: **merge** combines two trees into one, while **move** transforms one tree into another.

The last step is that the complementizer C merges with the TP we have just built. Since the TP does not undergo any subsequent movement operations, as indicated by the fact that its only feature is the category feature `T`, the C and TP merge straightforwardly, as in (44):

(44) (38f) **merge** (43) = ⟨([$_<$ ε [$_>$ Kat [$_<$ ε [$_>$ ε [$_<$ ε [$_<$ kicked Matt]]]]]], `C`)⟩

We will say that given C's status as the head of a (matrix) clause, the category feature `C` is privileged: a derivation is complete when we have a unary chain whose single FST has `C` as its sole remaining feature. We have thus successfully generated the sentence *Kat kicked Matt*.

3.3 Adding LF

We next move on to our revised MGs, in which PF and LF representations are derived simultaneously. In order to accomplish this, we will expand our class of movement licensee features: in addition to our "normal" movement licensee features (e.g., `-f`), we will also include *PF-only licensee features* (`-f`$_P$) and *LF-only licensee features* (`-f`$_L$). Movements triggered by normal (hereafter *PF+LF*) movement licensee features affect both PF and LF, making them overt scopal movements. Movements triggered by PF-only licensee features simulate reconstruction: since the movement does not impact LF, the constituent will scope at a position below its PF position. LF-only movement corresponds to QR, resulting in that constituent scoping *above* its PF position. There will remain only one type of movement licensor feature (`+f`), meaning that whether a constituent's movement affects PF, LF, or both will be determined strictly by the head's licensee feature. This is for convenience only: the analysis in this paper is equally compatible with a version in which it is the licensor that determines at which levels of syntactic representation a constituent moves, or in which the licensor and licensee work in tandem.

Since we are simultaneously building PF and LF trees, our FSTs will be replaced with **feature-specified tree pairs** (**FST2s**): triples consisting of a PF tree, an LF tree, and a string of features.

Chains will be tuples of FST2s. We will continue with our convention of using English orthography for PF representations; LF representations will use small-caps English orthography or, for silent heads, the name of the category of the head. However, these LF representations could just as well be treated as bundles of semantic features, or as lexical denotations; for our purposes it doesn't matter, and so we stick to small-caps orthography for ease of reading. Also as a matter of convenience, the three elements in each FST2 will be separated by line breaks rather than commas, with PF trees appearing on the top in red, and LF trees appearing in the middle in blue.

To see how our revised MGs work, we will go through a derivation of (8a) (*A technician inspected every plane*) on its inverse scope interpretation, where different planes may be inspected by different technicians. The lexical entries can be seen in (45):

(45)

a. $$\left\langle \left(\begin{array}{c} \text{every} \\ \textsc{every} \\ \texttt{=N D -sc}_{\texttt{L}} \end{array} \right) \right\rangle$$

b. $$\left\langle \left(\begin{array}{c} \text{plane} \\ \textsc{plane} \\ \texttt{N} \end{array} \right) \right\rangle$$

c. $$\left\langle \left(\begin{array}{c} \text{inspected} \\ \textsc{inspect} \\ \texttt{=D V} \end{array} \right) \right\rangle$$

d. $$\left\langle \left(\begin{array}{c} \varepsilon \\ \textsc{v} \\ \texttt{=V D= +sc v} \end{array} \right) \right\rangle$$

e. $$\left\langle \left(\begin{array}{c} \text{a} \\ \textsc{a} \\ \texttt{=N D -nom}_{\texttt{P}} \end{array} \right) \right\rangle$$

f. $$\left\langle \left(\begin{array}{c} \text{technician} \\ \textsc{technician} \\ \texttt{N} \end{array} \right) \right\rangle$$

g. $$\left\langle \left(\begin{array}{c} \varepsilon \\ \text{T} \\ \texttt{=v +nom T} \end{array} \right) \right\rangle$$

h. $$\left\langle \left(\begin{array}{c} \varepsilon \\ \text{C} \\ \texttt{=T C} \end{array} \right) \right\rangle$$

As mentioned previously, the red text in each lexical entry is the part inserted in the PF tree, and the blue text the part inserted in the LF tree. It should be clear from their phonetic representations which lexical items (45a–c) and (45e–f) are. (45d) is *v*, (45g) is T, and (45h) is C. Two noteworthy features of the entries in (45) are (i) that the movement licensee feature for the subject's determiner *a* is PF-only, meaning that the subject will overtly move to the usual spec-TP position, while scoping at its merge position in spec-*v*P; and (ii) that the object's determiner *every* has an LF-only licensee feature $\texttt{-sc}_{\texttt{L}}$ (for scope), with the corresponding licensor feature `+sc` appearing in the *v* head, meaning that the object will be pronounced in its merge position and take scope at the edge of the *v*P.

Let us now move on to the derivation. First, *every* merges with *plane*, checking *every*'s `=N` feature and *plane*'s `N` feature. This can be seen in (46):[19]

(46) $$\text{(45a) } \textbf{merge} \text{ (45b)} = \left\langle \left(\begin{array}{c} \text{[every plane]} \\ [\textsc{every plane}] \\ \texttt{D -sc}_{\texttt{L}} \end{array} \right) \right\rangle$$

Next, *every plane* merges in the complement of *inspected*. Since *every plane* has the LF-only movement feature $\texttt{-sc}_{\texttt{L}}$, it will be undergoing movement at LF, but not at PF. Now recall how movement was handled in the PF-only MG: rather than merging the phrase and then moving it, an empty "trace" was inserted from the get-go and the moving constituent was added to the chain to be tacked onto the tree later. But in this case, the PF portion of *every plane* will be merging with *inspected* and staying put there, while the LF portion must be held onto for later in order to enable the LF-only movement. So the PF half of this **merge** will look like one that occurs with no subsequent movement, and the LF half will look like a **merge** operation anticipating later movement.

[19]As in the original formulation of MGs in the previous section, the interior nodes of both PF and LF trees are < or > depending on the direction of the head of the constituent. We exclude these for ease of reading, though they are included in the derivation rules in Appendix A of the supplementary materials.

There is an additional complication involved in this LF movement. On traditional semantic analyses, traces denote free variables over individuals that saturate arguments at their merge positions. These variables are then lambda-abstracted over when the mover reaches its landing site, with the resulting predicate serving as an argument to the quantificational DP. But in order to do this, traces cannot simply be treated as empty nodes like we have been doing for PF: it is crucial that the traces have indices in order for lambda abstraction to successfully target the right free variable. Moreover, we want this to be deterministic: we do not want to conflate free variables by accidentally assigning distinct traces the same index.

The way this will be accomplished is as follows: we will posit a function "in", which is a bijective (one-to-one) function from movement licensee features to some finite subset of the set of natural numbers. When a phrase merges and will take scope via a feature $\texttt{-f}$ (or $\texttt{-f}_\text{L}$), the merge position is filled with a trace whose index is in($\texttt{-f}$) (or in($\texttt{-f}_\text{L}$)). Once the movement is complete, a node $\lambda_{\text{in}(\texttt{-f})}$ (or $\lambda_{\text{in}(\texttt{-f}_\text{L})}$) is inserted below the mover's landing site, lambda abstracting over the free variable.[20] This avoids a potential conflation of free variables due to an independent fact about MGs: **move** cannot take place if there are multiple possible movers with the same active licensee feature at the same time (the *Shortest Move Constraint*, Stabler 1997). Thus, if two movers utilize the same movement licensee feature, it must be the case that the second movement starts after the first movement is completed, at which point the latter's free variable will have already been lambda-abstracted over. There is therefore no risk of conflict.

With this in mind, the result of merging *inspected* with *every plane* can be seen in (47):

(47) (45c) **merge** (46) = $\left\langle \begin{pmatrix} \text{[inspected [every plane]]} \\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}] \\ \texttt{V} \end{pmatrix} ; \begin{pmatrix} \varepsilon \\ [\textsc{every plane}] \\ \texttt{-sc}_\text{L} \end{pmatrix} \right\rangle$

As promised, in the PF tree *every plane* merges with *inspected* and stays there. At LF, INSPECT combines with the trace $t_{\text{in}(\texttt{-sc}_\text{L})}$, since *every plane*'s new active feature is $\texttt{-sc}_\text{L}$. Since we need to hold onto the LF representation EVERY PLANE, we now have a binary chain; the PF portion of the non-primary FST2 is a placeholder empty tree ε.

Next, *v* merges with the newly-constructed VP, as in (48). Separately, *a* and *technician* merge together in preparation for the DP's merge into the specifier of *v*P. This can be seen in (49).

(48) (45d) **merge** (47) = $\left\langle \begin{pmatrix} [\varepsilon\ \text{[inspected [every plane]]]} \\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]] \\ \texttt{D= +sc v} \end{pmatrix} ; \begin{pmatrix} \varepsilon \\ [\textsc{every plane}] \\ \texttt{-sc}_\text{L} \end{pmatrix} \right\rangle$

(49) (45e) **merge** (45f) = $\left\langle \begin{pmatrix} \text{[a technician]} \\ [\textsc{a technician}] \\ \texttt{D -nom}_\text{P} \end{pmatrix} \right\rangle$

We now have the inverse from the situation in (47): because of the subject's PF-only licensee feature $\texttt{-nom}_\text{P}$, this time the DP *a technician* will be merging and staying put at LF, but undergoing movement at PF. The result is similarly inverted: an empty "trace" is inserted at PF, and A TECHNICIAN is merged at LF. The PF component is then added to our now ternary chain, with an empty placeholder ε in the LF position:

[20]In cases where a single constituent undergoes multiple LF movements, lambda abstraction takes place after each movement, with the entity argument of the ensuing predicate being saturated by the variable denoted by the trace left by the next LF movement.

(50) (48) **merge** (49) =

$$\left\langle \begin{pmatrix} [\varepsilon\ [\varepsilon\ [\text{inspected [every plane]]]]} \\ [[\textsc{a technician}]\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]]] \\ \texttt{+sc v} \end{pmatrix} ; \begin{pmatrix} \varepsilon \\ [\textsc{every plane}] \\ \texttt{-sc}_\text{L} \end{pmatrix} ; \begin{pmatrix} [\text{a technician}] \\ \varepsilon \\ \texttt{-nom}_\text{P} \end{pmatrix} \right\rangle$$

Now the primary FST2's active feature is the movement licensor `+sc`, which matches *every plane*'s LF-only $\texttt{-sc}_\text{L}$ licensee feature. As a result, EVERY PLANE is inserted in specifier position, with the lambda abstraction node $\lambda_{\text{in}(\texttt{-sc}_\text{L})}$ inserted below it.

(51) **move** (50) =

$$\left\langle \begin{pmatrix} [\varepsilon\ [\varepsilon\ [\text{inspected [every plane]]]]} \\ [[\textsc{every plane}]\ [\lambda_{\text{in}(\texttt{-sc}_\text{L})}\ [[\textsc{a technician}]\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]]]]] \\ \texttt{v} \end{pmatrix} ; \begin{pmatrix} [\text{a technician}] \\ \varepsilon \\ \texttt{-nom}_\text{P} \end{pmatrix} \right\rangle$$

Notice that nothing happens to the PF structure in this movement, since LF-only movement has no impact on the PF representation. The tense node T then merges with this completed *v*P, leading to the chain in (52):

(52) (45g) **merge** (51) =

$$\left\langle \begin{pmatrix} [\varepsilon\ [\varepsilon\ [\varepsilon\ [\text{inspected [every plane]]]]]} \\ [\text{T}\ [[\textsc{every plane}]\ [\lambda_{\text{in}(\texttt{-sc}_\text{L})}\ [[\textsc{a technician}]\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]]]]]] \\ \texttt{+nom T} \end{pmatrix} ; \begin{pmatrix} [\text{a technician}] \\ \varepsilon \\ \texttt{-nom}_\text{P} \end{pmatrix} \right\rangle$$

Since the primary FST2's active feature is the licensor `+nom`, *a technician*'s PF-only movement takes place by adding its PF component to spec-TP. The LF representation is unaffected by this PF-only movement.

(53) **move** (52) = $$\left\langle \begin{pmatrix} [[\text{a technician}]\ [\varepsilon\ [\varepsilon\ [\varepsilon\ [\text{inspected [every plane]]]]]]} \\ [\text{T}\ [[\textsc{every plane}]\ [\lambda_{\text{in}(\texttt{-sc}_\text{L})}\ [[\textsc{a technician}]\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]]]]]] \\ \texttt{T} \end{pmatrix} \right\rangle$$

And lastly, the finalized TP is merged with C, leading to a completed CP with an LF representation that leads to an inverse scope interpretation, as desired.

(54) (45h) **merge** (53) = $$\left\langle \begin{pmatrix} [\varepsilon\ [[\text{a technician}]\ [\varepsilon\ [\varepsilon\ [\varepsilon\ [\text{inspected [every plane]]]]]]]} \\ [\text{C}\ [\text{T}\ [[\textsc{every plane}]\ [\lambda_{\text{in}(\texttt{-sc}_\text{L})}\ [[\textsc{a technician}]\ [\textsc{v}\ [\textsc{inspect}\ t_{\text{in}(\texttt{-sc}_\text{L})}]]]]]]] \\ \texttt{C} \end{pmatrix} \right\rangle$$

We have thus successfully derived (8a) (*A technician inspected every plane*) on its inverse scope interpretation.

3.4 Derivation trees

Next we introduce *derivation trees*, which are a helpful way of visualizing syntactic derivations, and which will prove useful when we turn our attention to the MG parser. In a derivation tree, the leaf nodes (nodes without daughters) are lexical items—for our purposes represented as English orthography for overt items and as category labels for silent heads, plus the string of features—while the interior nodes are labeled **merge** and **move**. Since **move** is a unary operation (it takes a single chain and returns a different chain), interior nodes labeled **move** will be unary branching. By analogy, interior nodes labeled **merge** will be binary-branching, since **merge** is a binary operation. The linear order of nodes is irrelevant, since all information is encoded through hierarchical structure; however, we will order things in a way that maximally resembles overt word order.

The derivation tree for our derivation of the inverse scope interpretation of (8a) (*A technician inspected every plane*) can be seen in Figure 2. The derivation tree should be read "bottom up": first, *every* merges with *plane*, with the result merging with *inspected*, etc. Thus, the lower of the two **move** nodes represents the LF-only movement of *every plane*, since that happened first, while the higher of the two represents the PF-only movement of *a technician*.

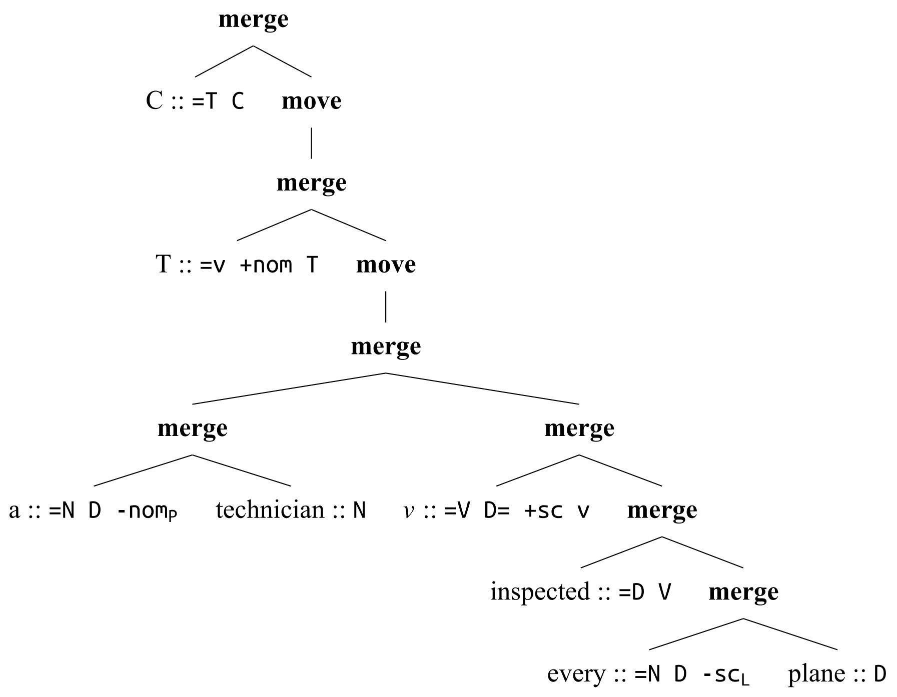


Figure 2: Derivation tree for inverse scope reading of *A technician inspected every plane*.

For the rest of this paper, we will make three modifications to our derivation trees in order to facilitate reading. The first is that we will not show the string of features for each lexical item, and will only provide the name of that lexical item. The second is that **merge** and **move** nodes will be replaced with the traditional labels of the constituents created by means of those operations, e.g., the lowest **merge** node will be replaced with DP, and the highest **move** node will be replaced with TP. The third is that movement arrows will be included, starting at the root of moved phrases and going to the relevant movement node, roughly corresponding to the landing site.[21] Red arrows will indicate PF-only movement, blue arrows LF-only movement, and black arrows PF+LF movement. A crucial difference between these movement arrows and traditional movement arrows will be that for ours, the origin of the arrow will always be at the root of the moved constituent: if a constituent moves twice, the arrow for the second movement will not be from the landing site of the first movement to the landing site of the second movement, but rather from the root of the moved constituent to the landing site of the second movement. The simplified derivation tree for our inverse scope derivation of *A technician inspected every plane* can be seen in Figure 3.

[21]The *root* of a (sub)tree is the single node in that (sub)tree that dominates all other nodes.

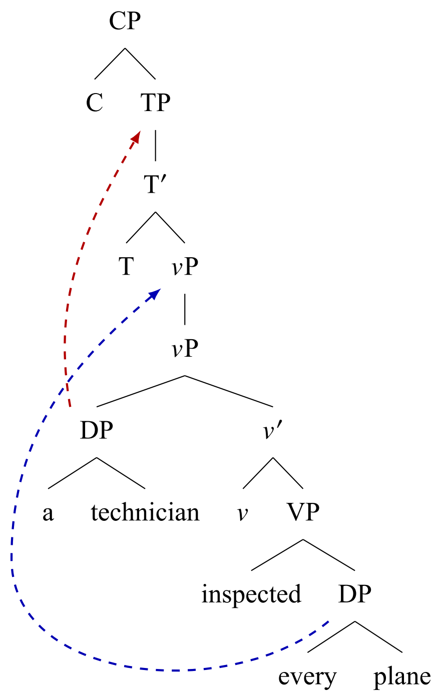


Figure 3: Simplified derivation tree for inverse scope reading of *A technician inspected every plane*.

Before moving on to the MG parser, one more thing is worth noting about derivation trees. In introducing derivation trees, we said that the leaves represented lexical items, while the interior nodes represented derivation steps (**merge** or **move**). But there is another, equally useful way of looking at things: each node corresponds to a chain that we see during the course of the derivation. For the leaf nodes, the chains are the lexical entries for the corresponding lexical items. For interior nodes, the chain is the one that is built by the corresponding operation. Thus, the **merge** node that immediately dominates *every* and *plane* (labeled DP in Figure 3) corresponds to the chain built by the **merge** operation in (46), while the lower **move** node (labeled as the higher *v*P in Figure 3) corresponds to the chain resulting from LF-only movement of *every plane* in (51). This observation will prove especially useful when discussing the MG parser, to which we now turn.

4 A semi-formal introduction to MG parsing

We now turn to the top-down MG parser, which essentially does the syntactic derivation of a sentence in reverse, but in a way that is sensitive to lexical items' linear order (at PF). This parser will be used to define our metric for scope processing difficulty in §5. In this section in particular, we will discuss things only in very informal terms; a complete formal definition of the parser can be found in the appendices. We start in §4.1 with a cautionary note, aimed especially toward those unfamiliar with work on formal parsing, about what the parser does and does not do. In §4.2 we illustrate how the parser works by parsing our derivation for (8a) (*A technician inspected every plane*) from the previous section. Finally, in §4.3 we will introduce *annotated derivation trees*, which are a convenient way of using derivation trees in order to show the path the parser takes.

4.1 Prelude to the parser: What's in a parser?

In the next subsection, we will introduce the formal parser for MGs used in defining and testing our scope processing difficulty metric. However, for the sake of those unfamiliar with work on formal parsing, a note is warranted on what is, and more importantly what is not, included in the parser.

At its core, the job of any syntactic parser—including the human parser—is to take a string S and determine a well-formed syntactic structure (or derivation) J such that the string yield for J is S.[22] An implementation of such a parser can be usefully thought of as consisting of two modules. The first is a mechanism that allows one to make predictions about what the syntactic structure (or derivation) looks like, to keep track of the precise ramifications of those predictions, and to scan the input string to determine if those predictions are correct. The second is an algorithm that determines the path of the parse by deciding what prediction and confirmation steps to actually take.

By way of illustration, consider the case of garden path sentences like the famous (55):

(55) The horse raced past the barn fell.

The human parser's first inclination is to parse *raced* as the past tense of *race*, but this leads to a crash: *The horse raced past the barn* is a complete sentence, and there is no place to fit *fell*. Instead, one can only successfully parse (55) by treating *raced past the barn* as a reduced relative clause, so that the structure of (55) is similar to that of the much more easily processed (56).

(56) The horse that was raced past the barn fell.

The garden path phenomenon can plausibly be traced in large part to the second module mentioned above: when parsing (55), the human parser is highly prone to making the wrong guesses, and has difficulty determining what guesses are the right ones to make. As a result, once one is "let in on the joke" and informed of the correct parse—perhaps by means of a paraphrase like (56)—there is often little difficulty in determining the structure of the original string.

In contrast, consider again the case of center-embedding in (1b), repeated below:

(1b) The mouse that the cat that the dog that the barber owned bit ate liked cheese.

The difficulty involved in processing (1b) seems to be of an altogether different sort from that involved in processing (55). Even upon being told that (1b) has such-and-such structure and being given a more easily parsed paraphrase like (1a), actually navigating the structure of (1b) is still exceedingly difficult without pen and paper. This suggests that the heart of the difficulty of (1b) lies in the first module above: what makes (1b) hard to process is not (or not just) an inclination to make the wrong guesses, but rather a problem of keeping track of all of the information that the prediction-making mechanism has to store, even when making all of the right guesses.

We bring this up because the parser introduced in the next subsection and used throughout this paper includes the first module and excludes the second: it is a mechanism for prediction-making and -confirming, devoid of any algorithm that tells us what predictions (not) to make for a given parse. Thus, whenever we go step-by-step through a parse, this parse will be one in which only correct predictions are made, thereby abstracting away from the issue of how the human parser actually determines which predictions to make. In light of the discussion above, this is not a theory-neutral abstraction, but an empirically loaded one: it comes with the prediction that in terms of processing difficulty, inverse scope looks qualitatively more like center-embedding than like garden

[22]The *string yield* for a given syntactic structure is the string that is its output, i.e., the "pronounced" string.

paths. We think that this is correct: Anderson (2004) provides ample evidence that inverse scope is difficult to process even with an appropriately biasing context, and the back-and-forth in the theoretical literature shows that long QR can be difficult to get even for those who know better than most what such an interpretation would look like.

4.2 The parser

We now turn to the parser itself. Our derivation made use of *chains*, which were ordered tuples of *feature-specified tree pairs*. By analogy, our parse will make use of *parse items*, which are ordered tuples of *doubly-addressed feature strings* (AFS2s). There will be a one-to-one correspondence between parse items and chains, and between AFS2s and FST2s: each chain in a derivation will have a corresponding parse item in a parse, and within a given chain, each FST2 will have a corresponding AFS2 in the corresponding parse item.

We start by defining AFS2s. An AFS2 can be thought of as a prediction about some substructure of the PF and LF trees: roughly, it is a prediction that there is a subtree whose root occupies a particular position at PF and a particular position at LF, and whose head has a particular string of remaining features at the relevant point in the derivation. Much like how FST2s were ordered triples consisting of a PF tree, an LF tree, and a string of features, AFS2s will be ordered triples consisting of a *PF address*, an *LF address*, and a string of features. In the parser defined in the appendices, these addresses are *gorn addresses*, which are formally defined objects that pinpoint a particular node in a tree. However, for the informal purposes of understanding our scope processing metric, we do not need such fine-grained information; therefore, the "addresses" in our AFSs will be replaced with check marks indicating that the parser has determined the relevant location.

Since parse items are to AFS2s what chains are to FST2s, parse items are naturally tuples of AFS2s. The role of AFS2s in parse items is similar as well: the first, *primary AFS2* provides information about the main part of the tree we are parsing, while non-primary AFS2s provide information about moving constituents whose existence has been predicted.

Since a parse essentially performs the derivation in reverse, we start by predicting a completed CP. Thus, the predicted PF and LF addresses for the root of our subtree will simply be the root of the CP as a whole, while the feature string will consist only of the category feature C, since that is the only feature remaining at the end of a derivation (cf. (54)). Thus, given that all relevant positional information is known at this point in the derivation, our starting parse item will be ⟨(✓, ✓, C)⟩; note that we continue our color-coding, with red indicating PF addresses and blue LF addresses. We will also add further helpful notation: for each AFS2, we will include as a superscript the label of the constituent being predicted. Thus, we will notate our starting parse item as $\langle(✓, ✓, \texttt{C})^{\text{CP}}\rangle$. Note that this superscript should be read not as a part of the actual parser, but rather as a guide.

Each step in the parse of a sentence is of one of three categories: **unmerge**, which "undoes" a **merge** operation; **unmove**, which does the same for a **move**; and **scan**, which checks to see if the next word in the input string meets the conditions predicted by the parse. (In the case of a phonetically empty lexical item, the scan is always successful.) More generally, the parser can be thought of as predicting that a certain sequence of operations has taken place, and then confirming those predictions by scanning lexical items in their linear order.

Since the last step in our derivation was to **merge** C and TP, this is the first step we undo via **unmerge**. In a derivation, **merge** combines two chains into a single chain, deleting selector and category features; conversely, in a parse, **unmerge** splits a single parse item into two parse items, adding selector and category features. Since we are predicting that C selected TP to its right via `=T`, we immediately know where the roots of the two newly predicted sub-trees are: we know that C is the left daughter of the CP root at both PF and LF, and TP is its right daughter. The result of this first step of **unmerge** can be seen in (57); note the analogy (including identity of feature strings) between the newly introduced parse items and the chains fed into **merge** in (54).[23]

(57)
$$\frac{\langle(\checkmark,\ \checkmark,\ \texttt{C})^{\mathrm{CP}}\rangle}{\langle(\checkmark,\ \checkmark,\ \texttt{=T C})^{\mathrm{C}}\rangle\quad\langle(\checkmark,\ \checkmark,\ \texttt{T})^{\mathrm{TP}}\rangle}\ \textbf{unmerge}$$[24]

We now have multiple parse items to play with. By rule, we can only operate on the first-listed remaining parse item, i.e., $\langle(\checkmark,\ \checkmark,\ \texttt{=T C})^{\mathrm{C}}\rangle$. But what determines the order of parse items? The answer is linear order within the PF tree: the first-listed parse item is the one containing the leftmost PF address, the second-listed is the one containing the next leftmost PF address, etc. This is because lexical items must be scanned *in left-to-right order*. Thus, the parser will always take the fastest route that will allow it to scan from left to right. Since the input to the parser is a PF string, and not an LF string, what matters is the left-to-right order of PF addresses, not LF addresses. (In this case these happen not to conflict.) Since C is predicted to be to the left of TP at PF, this means that the parse item for C is listed first.

The feature string `=T C` matches the feature string for the lexical item C as indicated in (45h). Moreover, since the phonetic representation for C is the empty string ε, we can **scan** it "for free": scanning is automatically successful. We thus scan C, meaning that its parse item is deleted.

(58)
$$\frac{\langle(\checkmark,\ \checkmark,\ \texttt{=T C})^{\mathrm{C}}\rangle\quad\langle(\checkmark,\ \checkmark,\ \texttt{T})^{\mathrm{TP}}\rangle}{\langle(\checkmark,\ \checkmark,\ \texttt{T})^{\mathrm{TP}}\rangle}\ \textbf{scan}\ \mathrm{C}$$

Our remaining parse item is the predicted TP, the right daughter of CP at both PF and LF. As seen in (53), the last derivation step in the building of this TP was **move**, which PF-moved the subject to spec-TP. This operation took a single chain with multiple FSTs—one for the "main" tree, one for the mover—and replaced it with a single chain with a single FST, the result of tacking the mover onto the PF tree at spec-TP, simultaneously deleting T's `+nom` feature and the determiner's `-nom`$_{\mathrm{P}}$ feature. In undoing this **move** operation, we do more or less the same thing in reverse: we take a single parse item with a single AFS2, and return a single parse item with two AFS2s, a primary one for the main tree with `+nom` added as a feature, and a secondary one for the mover with `-nom`$_{\mathrm{P}}$ added as a feature. Because of how movement works—the mover always moves to a newly created specifier—we know that at PF, the DP mover is the left daughter of TP, and the rest of the tree is the right daughter of TP.

But what about at LF? The PF-only movement did not change anything about the LF structure. Therefore, nothing has changed in the main (TP) LF tree, meaning we still know where its root is. But as for the subject, all the parser currently knows about it is that it PF-moved to spec-TP by means of a `-nom`$_{\mathrm{P}}$ feature. While this is enough to tell us where the subject sits at PF, an LF address

[23]We operate within a "parsing as deduction" framework (Pereira & Warren 1983), which is commonly used in work on syntactic parsing. For us, parses are deductions in which the sole axiom is the initial parse item, the inference rules are the parse rules, and the goal for a successful deduction is the elimination of all parse items and the scanning of the whole input string.

[24]Naming the parse rule **unmerge** is somewhat misleading because there are multiple **unmerge** parse rules (and likewise for **unmove**), as shown in Appendix B. In the body of the paper we will continue to use the simplified names.

cannot yet be determined: we do not yet know where the subject moved *from*. More generally, we do not know where a constituent sits at LF until we have undone some LF-relevant operation, whether that be **merge**, LF-**move**, or PF+LF-**move**. We will therefore use a question mark for the subject's LF address, indicating that we do not yet know where the subject sits at LF:

(59)
$$\frac{\langle(\checkmark,\ \checkmark,\ \texttt{T})^{\mathrm{TP}}\rangle}{\langle(\checkmark,\ \checkmark,\ \texttt{+nom T})^{\mathrm{T}'};\ (\checkmark,\ ?,\ \texttt{-nom}_{\texttt{P}})^{\mathrm{DP}}\rangle}\ \textbf{unmove}$$

Note once again the analogy between derivation chains and parse items. The input to **move** in (53)—that is, the chain built in (52)—was a single chain with two FST2s, the primary having the feature string $\texttt{+nom T}$, and the secondary (mover) having the feature string $\texttt{-nom}_{\texttt{P}}$. Meanwhile, the parse item resulting from **unmove** also contains two AFS2s, with both the primary and secondary having the same feature strings as their corresponding FST2s. While we will not continue to hammer on this point, it is worth emphasizing that this correspondence between chains and parse items will hold at every step of the parse; the distrusting reader is left to confirm this on their own.

Since we currently have a predicted T′, the next derivation step we undo is the **merge** of T and *v*P that built this T′. As before, this **unmerge** takes our single parse item and returns two parse items. Since the PF and LF addresses for T′ are known, the PF and LF addresses for T (left daughter) and *v*P (right daughter) are also immediately recoverable from this parse operation, though the DP mover's LF address is still a mystery—in fact, the DP's AFS2 goes unchanged since the DP was uninvolved in this derivation operation. This **unmerge** step can be seen in (60):

(60)
$$\frac{\langle(\checkmark,\ \checkmark,\ \texttt{+nom T})^{\mathrm{T}'};\ (\checkmark,\ ?,\ \texttt{-nom}_{\texttt{P}})^{\mathrm{DP}}\rangle}{\langle(\checkmark,\ \checkmark,\ \texttt{v})^{v\mathrm{P}};\ (\checkmark,\ ?,\ \texttt{-nom}_{\texttt{P}})^{\mathrm{DP}}\rangle\quad\langle(\checkmark,\ \checkmark,\ \texttt{=v +nom T})^{\mathrm{T}}\rangle}\ \textbf{unmerge}$$

Two things are worth noting about this step. First, note that the DP's AFS2 is now hitched to *v*P, and not T. The reason for this is that movers' AFS2s always ride with the AFS2s of the constituents within which they were merged, in order to enable the ultimate **unmerge** of that mover. Since the subject was merged *v*P-internally, its AFS2 thus rides with the *v*P. Second, notice that even though T is to the left of *v*P at PF, the latter's parse item is listed first, and not the former's. This is because of all of the known PF addresses, the leftmost is not T's, but the subject's, meaning that the parse item containing the subject's AFS2 is the one listed first.

The last derivation step in the building of the *v*P was the LF-only **move** of the object to the edge of *v*P, meaning that this is what we next undo. Recall that when we undid the subject's PF-only movement, its final PF position was immediately recoverable, but its LF position was not. The inverse happens when we undo an LF-only movement. Because movement puts movers in specifier positions, once we undo the LF-only movement of the object we immediately know that at LF the object sits at the edge of *v*P. But since all the parser knows about the object at this point is that it LF-moved to the edge of *v*P, the parser cannot yet determine the object's PF location; this will not be known until some PF-relevant operation is undone. The result can be seen in (61):

(61)
$$\frac{\langle(\checkmark,\ \checkmark,\ \texttt{v})^{v\mathrm{P}};\ (\checkmark,\ ?,\ \texttt{-nom}_{\texttt{P}})^{\mathrm{DP}}\rangle\quad\langle(\checkmark,\ \checkmark,\ \texttt{=v +nom T})^{\mathrm{T}}\rangle}{\langle(\checkmark,\ \checkmark,\ \texttt{+sc v})^{v\mathrm{P}};\ (\checkmark,\ ?,\ \texttt{-nom}_{\texttt{P}})^{\mathrm{DP}};\ (?,\ \checkmark,\ \texttt{-sc}_{\texttt{L}})^{\mathrm{DP}}\rangle\quad\langle(\checkmark,\ \checkmark,\ \texttt{=v +nom T})^{\mathrm{T}}\rangle}\ \textbf{unmove}$$

As before, the **unmove** replaces a parse item with a single other parse item, introducing a new AFS2 for the newly predicted mover, and introducing a $\texttt{+sc}$ feature for the predicted *v* and a $\texttt{-sc}_{\texttt{L}}$ feature for the predicted mover. There are also now two unknown addresses: the subject's LF address and the object's PF address.

The next step that we undo is the merging of the subject into spec-*v*P. As always, this splits the first-listed parse item into two parse items; this time, the subject's AFS2 "breaks away" into its own parse item, meaning we now have three parse items in play. In addition, this **unmerge** finally gives the parser enough information to infer where the subject sits at LF: it was merged in spec-*v*P and did not move from there. Thus, the LF "address" can be switched from ? to ✓. This can be seen in (62); parse items are split onto two lines for readability:

(62)
$$\begin{array}{rl}
\langle(\checkmark, \checkmark, \texttt{+sc v})^{vP}; (\checkmark, ?, \texttt{-nom}_{\texttt{P}})^{DP}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \\ \hline
\langle(\checkmark, \checkmark, \texttt{D -nom}_{\texttt{P}})^{DP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \textbf{unmerge} \\
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle &
\end{array}$$

Notice that the parse items have also been reordered: since the subject has its own parse item with the leftmost PF address (spec-TP), that is now listed first, followed by T, followed by *v′*. (Since the object's PF address is unknown, it does not contribute to determining the order of parse items.)

Since the subject DP is the first-listed parse item, our next step in the parse is to undo the step that built that DP: namely, the **merge** of *a* and *technician*. This is straightforward:

(63)
$$\begin{array}{rl}
\langle(\checkmark, \checkmark, \texttt{D -nom}_{\texttt{P}})^{DP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \\
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \\ \hline
\langle(\checkmark, \checkmark, \texttt{=N D -nom}_{\texttt{P}})^{D}\rangle\ \langle(\checkmark, \checkmark, \texttt{N})^{NP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \textbf{unmerge} \\
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle &
\end{array}$$

Next, the parser can **scan** *a* because the next word is pronounced *a*, and the feature string of the parse item matches that of the lexical entry for *a*. Then, *technician* can be scanned for the same reason. Finally, since T is silent, it can also be scanned, since once again the feature string for T's parse item matches that of the lexical entry for T.

(64)
$$\begin{array}{rl}
\langle(\checkmark, \checkmark, \texttt{=N D -nom}_{\texttt{P}})^{D}\rangle\ \langle(\checkmark, \checkmark, \texttt{N})^{NP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \\
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \\ \hline
\langle(\checkmark, \checkmark, \texttt{N})^{NP}\rangle\ \langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle & \textbf{scan } a \\
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \\ \hline
\langle(\checkmark, \checkmark, \texttt{=v +nom T})^{T}\rangle\ \langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \textbf{scan } \textit{technician} \\ \hline
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \textbf{scan } \text{T}
\end{array}$$

This brings us back to a single parse item, which contains the AFS2s for the predicted *v′* and the object DP. Next we undo the step that built this *v′*, namely, the **merge** of *v* and VP. Notice that the object's AFS2 goes along with the VP's, since the object was obviously merged inside the VP. The silent *v* head can then be scanned:

(65)
$$\begin{array}{rl}
\langle(\checkmark, \checkmark, \texttt{D= +sc v})^{v'}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \\ \hline
\langle(\checkmark, \checkmark, \texttt{=V D= +sc v})^{v}\rangle\ \langle(\checkmark, \checkmark, \texttt{V})^{VP}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \textbf{unmerge} \\ \hline
\langle(\checkmark, \checkmark, \texttt{V})^{VP}; (?, \checkmark, \texttt{-sc}_{\texttt{L}})^{DP}\rangle & \textbf{scan } v
\end{array}$$

Next, the step that built the VP—the **merge** of the verb and object DP—can be undone. As before, this **unmerge** gives the object AFS2 its own parse item. Moreover, this **unmerge** finally gives the parser enough information to determine where the object sits at PF: it was merged as the right daughter of VP and did not move from there. Thus, the PF "address" is switched from ? to ✓. The verb *inspected* can then be scanned. This is followed by the unmerging and scanning of *every* and *plane*, which goes as expected. This is all shown in (66).

(66)
$$\begin{array}{r l}
\langle(\checkmark,\ \checkmark,\ \texttt{V})^{\mathrm{VP}};\ (?,\ \checkmark,\ \texttt{-sc}_{\texttt{L}})^{\mathrm{DP}}\rangle & \\ \hline
\langle(\checkmark,\ \checkmark,\ \texttt{=D V})^{\mathrm{V}}\rangle\ \ \langle(\checkmark,\ \checkmark,\ \texttt{D -sc}_{\texttt{L}})^{\mathrm{DP}}\rangle & \textbf{unmerge} \\ \hline
\langle(\checkmark,\ \checkmark,\ \texttt{D -sc}_{\texttt{L}})^{\mathrm{DP}}\rangle & \textbf{scan}\ \textit{inspected} \\ \hline
\langle(\checkmark,\ \checkmark,\ \texttt{=N D -sc}_{\texttt{L}})^{\mathrm{D}}\rangle\ \ \langle(\checkmark,\ \checkmark,\ \texttt{N})^{\mathrm{NP}}\rangle & \textbf{unmerge} \\ \hline
\langle(\checkmark,\ \checkmark,\ \texttt{N})^{\mathrm{NP}}\rangle & \textbf{scan}\ \textit{every} \\ \hline
 & \textbf{scan}\ \textit{plane}
\end{array}$$

We have successfully completed the parse: the input string (*A technician inspected every plane*) has been fully scanned, and there are no more parse items remaining.

4.3 Annotated derivation trees

In the previous section we introduced *derivation trees*, which are a way of representing syntactic derivations. We will now introduce *annotated derivation trees*, which are a convenient way of representing both the derivation and the path the parser takes in parsing the derived structure. As previously noted, there is a one-to-one relationship between the chains that arise in a derivation and the parse items arising in its corresponding parse. And as also noted, there is a one-to-one relationship between the nodes in the derivation tree and the chains in a derivation: leaf nodes correspond to lexical chains, **merge** nodes correspond to chains built by **merge** operations, and **move** nodes correspond to chains built by **move** operations. Therefore, there is an additional one-to-one relationship between the nodes in a derivation tree and the parse items in the parse: the leaf nodes correspond to those parse items that are discharged (removed) by **scan** operations, **merge** nodes correspond to parse items that are discharged by **unmerge** operations, and **move** nodes correspond to parse items that are discharged by **unmove** operations.

With this in mind, we can illustrate the path of the parse by assigning each node in the derivation tree an *index*, which indicates at what step in the derivation the corresponding parse item is introduced, and an *outdex*, indicating the step in the derivation at which that parse item is discharged. The annotated derivation tree for our derivation/parse of the inverse scope of (8a) can be seen in Figure 4; indices are on the top left, and outdices are on the bottom right.

Take, for example, the CP node, which is a **merge** node, as is recognizable from the fact that it is binary branching. The CP node corresponds to the very first parse item, $\langle(\checkmark,\ \checkmark,\ \texttt{C})^{\mathrm{CP}}\rangle$, which is introduced in the very first step and immediately discharged via **unmerge**. This can be contrasted with the node labeled T, which is the parse item that is eventually discharged when T is scanned; this parse item is introduced in the fifth step (when T′ is unmerged, cf. (60)), and is not discharged via **scan** until six steps later. More generally, a parse item's *tenure* is the number of steps between when it is introduced and when it is discharged, which can be gleaned from the annotated derivation tree by subtracting its corresponding node's index from its outdex. This notion of tenure has featured prominently in work on the relation between processing and Minimalist parsing, as the tenure of a parse item could reasonably be thought to correspond to how long a given prediction must be

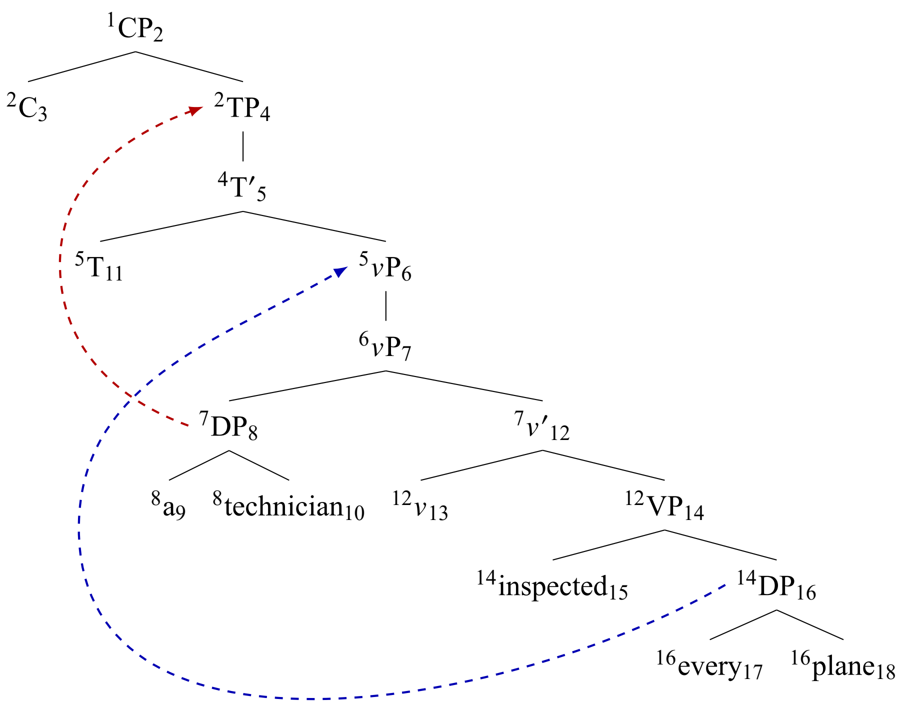


Figure 4: Annotated derivation tree for inverse scope reading of *A technician inspected every plane*.

retained in the human parser's working memory.[25] While our analysis of scope processing will not be built specifically around tenure, it will similarly be defined based on the nature and duration of information storage during the course of a syntactic parse. We are now in a position where we can define this novel scope processing difficulty metric.

5 A new metric for scope processing difficulty

It is now time to offer our analysis of scope processing difficulty. In short, the idea is this: depending on the operations that take place during a derivation, at various points in the parse there may be constituents whose existence has been predicted, but whose location at PF or LF cannot yet be determined. We will argue that processing difficulty is correlated with how many of these constituents there are during the course of a parse, as well as how long (in terms of number of parse steps) it takes until their location at both PF and LF is determined. We will see that this principle generates the right predictions for all of the scope processing observations discussed in §2.

[25]Kobele et al. (2013) were the first to discuss and use tenure in MG parsing. For non-MG predecessors to tenure-based analyses of processing difficulty, see Joshi 1990, Rambow & Joshi 1994.

5.1 The SLD Principle

5.1.1 Defining the SLD Principle

Recall that in our parse of the inverse-scope reading of (8a) (*A technician inspected every plane*), there were a number of steps in which the PF "address" assigned to *every plane* was ?, indicating that it could not yet be determined. Once *every plane* was unmerged, its PF location as the right daughter of VP could be determined. We can summarize the reason for this as follows. With our expanded class of movement operations, we now have four derivation operations in total: **merge**, PF+LF-**move**, PF-**move**, and LF-**move**. Naturally, only the first three of these are PF-relevant operations, in the sense that they impact the PF representation of the sentence; as far as PF is concerned, all LF-only movement does is check features. Moreover, it is always the case that the last PF-relevant operation determines the final PF position of a constituent. Thus, in undoing the derivation during the course of the parse, one does not know where a constituent sits at PF until the first undoing of a PF-relevant operation, since this undoes the last PF-relevant derivation step. In the case of *every plane*, this last PF-relevant derivation step was its **merge** with *inspected*. The result was that the AFS2 for *every plane* had an unknown PF address for those parse steps between when it was first predicted (after **unmove**) and when the **unmerge** was completed. The same was true of the LF address for the subject *a technician*: since the first operation that was undone (the PF-only **move** to spec-TP) was not LF-relevant, the position of *a technician* at LF could not be determined until some LF-relevant operation—namely, its initial **merge** in spec-*v*P—was undone. As a result, *a technician*'s LF address was unknown until this **unmerge** took place.

According to our scope processing difficulty metric, the difficulty of a parse is tied to how long (in terms of number of parse steps) various moving constituents are predicted to exist, but with information missing about their PF or LF locations. We therefore need a way to find this number of steps for a given mover—what we will call its **location differential**. Luckily, this is easy when looking at the parse: for a given mover, we count the number of tokens of ? throughout the parse. Take, for example, *a technician*. The existence of this constituent is predicted in step 4 of the parse, corresponding to the second line of (59). At this point, its PF location (spec-TP) is known, but its location at LF is not. This continues to be the case until step 7, the second line of (62), at which point the LF location of *a technician* is finally determined to be spec-*v*P. Thus, since there are 3 steps during which *a technician* has been predicted to exist without a determined LF location, its location differential is 3. Similarly, *every plane* has an unknown PF location from when it is introduced in step 6 (second line of (61)) until it is unmerged in step 14 (second line of (66)), meaning its location differential is 8.

In order to compare the relative scope processing difficulty of two parses, for each parse we find the sum of the location differentials for all the movers involved—what we will call its **summed location differential (SLD)**—and then we compare the sums for the two parses. We call this the **SLD Principle**, defined in (67):

(67) **SLD Principle:**
Parse *A* incurs a greater processing cost than Parse *B* if *A*'s SLD is greater than *B*'s.

Importantly, the SLD Principle only provides us with an *ordinal scale* for ranking parses in terms of processing cost: it tells us which parses are costlier than others, but not how significant the difference is in cost between two parses. Thus, while SLD assigns a number to a given parse, and while the comparison of these numbers serves as our metric for processing difficulty, the relative size of the difference between two SLDs should not necessarily be construed as an indication of the relative difference in processing difficulty of the two parses. Of course, there may indeed be a correlation in this regard, but this is an empirical question over and above the one we are attempting to address in this paper, and one that ought to be addressed via experimentation before a proper computational account can be provided.

5.1.2 Derivation tree geometry as a shortcut to location differentials

In the rest of this paper, we will use a bit of a shortcut to find the location differentials of the movers in a given parse, by using the parse's annotated derivation tree. To see how this works, suppose that there is some mover *m* whose final movement is an LF-only movement. Because this LF-only movement will be the first to be undone in the parse, after this **unmove** happens the LF position of *m* will be known, but the PF position will not, for the reasons stated above. The location differential for *m* will therefore be the number of parse steps from when it is first predicted (unmoved) to when it is assigned a PF position, and we will know the PF position for *m* after undoing some PF-relevant operation for *m*. Now because of another fact about our extended MGs—namely, that no constituent can undergo both LF-only and PF-only movement in the same derivation (tantamount to undergoing both QR and reconstruction)—we know that *m*'s PF position can only be determined by undoing a **merge** or PF+LF-**move** of *m*. Similar logic entails that if *m*'s last movement is PF-only, its location differential will be the number of steps between when it is first predicted and when a **merge** or PF+LF-**move** of *m* is undone, since this is how long its LF position will be unknown. Finally, if *m*'s last movement is a PF+LF movement, the location differential will be zero, since we will know both the PF and LF locations of *m* as soon as *m*'s existence is predicted. (For instance, if *m*'s final movement is a PF+LF movement to spec-TP, when we undo this movement we immediately know that *m* is in spec-TP at both PF and LF.) Thus, no matter what, the following principle holds: *for a given mover m, m's location differential is the number of steps from when it is first predicted to when it is first either unmerged or PF+LF-unmoved.*

This can easily be spotted by looking at the annotated derivation tree. Recall that the interior nodes of a derivation tree represent operations in the derivation, and the outdex of an interior node represents the step in the parse when that operation is undone. Therefore, the existence of a mover is first predicted at the step corresponding to the outdex of its final movement operation—that is, the highest move with an arrow (of any color) from the mover. Meanwhile, the first step at which both the PF and LF locations of a mover are known will be the outdex either of that mover's immediately dominating **merge** node—corresponding to when that mover is unmerged—or of the highest PF+LF-**move** node for that constituent, if that constituent ever undergoes PF+LF movement.

Take, for example, the annotated derivation tree in Figure 4 above for our parse of (8a). The existence of *a technician* is predicted at step 4, which is the oudex of its highest **move** node (labeled TP). Since *a technician* never undergoes PF+LF movement, its LF location is not known until it is unmerged in step 7, which is the outdex of the DP's immediately dominating **merge** node (the lower *v*P). Thus, *a technician*'s location differential is $7 - 4 = 3$, as before. Similarly, the existence of *every plane* is predicted at step 6, the outdex of its highest **move** node (the higher *v*P), and again since *every plane* never undergoes PF+LF movement its PF location is not known until step 14, the outdex of the immediately dominating **merge** node (VP). *every plane*'s location differential is therefore $14 - 6 = 8$, again as before. We thus arrive at an SLD of $8 + 3 = 11$ for this parse.

Now that the SLD Principle has been clearly defined, we will show how it garners the right results for all of the cases discussed in §2. We will only show those details about derivations and parses that are necessary to illustrate how the SLD Principle derives the correct results; full details on the derivations and parses used can be found in the appendices.

5.2 Example 1: Subject > Object

The first scope preference observation we will account for via the SLD Principle will be the preference for subjects to scope over direct objects in simple transitives (Kurtzman & MacDonald 1993, Tunstall 1998, Anderson 2004). We have already seen what an inverse scope derivation and parse for a simple transitive look like for *A technician inspected every plane*, but what about a surface scope interpretation? We will assume, following Heim & Kratzer (1998) and others, that due to a type mismatch quantified objects have to undergo LF movement regardless of scope configuration, meaning that the LF-only movement of *every plane* will remain (but see §5.4). The way we will derive surface scope will be by replacing the PF-only movement of *a technician* with a PF+LF movement, so that the subject takes scope in spec-TP, rather than in its merge position. This is accomplished by swapping out the determiner *a*'s PF-only licensee `-nom`$_{\mathtt{P}}$ feature for a PF+LF licensee `-nom` feature. The derived PF tree is the same as before; the derived LF tree is as in (68), leading to a surface scope reading:

(68) [C [[A TECHNICIAN] [$\lambda_{\text{in}(\texttt{-nom})}$ [T [[EVERY PLANE] [$\lambda_{\text{in}(\texttt{-sc}_{\text{L}})}$ [$t_{\text{in}(\texttt{-nom})}$ [V [INSPECT $t_{\text{in}(\texttt{-sc}_{\text{L}})}$]]]]]]]]

Since replacing PF-only movement with PF+LF movement has no effect on PF word order, the path that the parser takes through our derivation tree will be the same as before, meaning that the indices and outdices of the nodes will be the same as in Figure 4. In other words, as far as our annotated derivation tree is concerned, the only result we see is that the red movement arrow of *a technician* is replaced with a black one, indicating PF+LF movement. This can be seen in Figure 5.

Now that we have our two parses, it is time to compare SLDs. Recall that for the inverse scope derivation in Figure 4, *a technician* had a location differential of 3, while *every plane* had a location differential of 8. Thus, the SLD for that parse was $8 + 3 = 11$. For the surface scope derivation in Figure 5, the location differential of *every plane* is still 8: *every plane* is still predicted at step 6 (the outdex of the higher *v*P node), and it is still not assigned a PF address until step 14 (the outdex of the immediately dominating **merge** node). But things are different for the subject: since the only movement is a PF+LF movement, the instant this operation is undone we know precisely where *a technician* will sit at both PF and LF, namely spec-TP. Thus, *a technician* has a location differential of 0 instead of 3, so the parse as a whole has an SLD of 8 instead of 11. Since the SLD Principle states that whichever parse has the lowest SLD is the easiest to process, we correctly predict that surface scope is less costly than inverse scope here.

Before moving on, note that there is another way we could have derived inverse scope: rather than the subject scoping in its merge position with the object's obligatory movement leapfrogging it, the subject could have scoped in its landing site (spec-TP), with the object undergoing additional QR to a position above this landing site. The same could be done for the cyclic QR cases discussed in §5.4. For brevity's sake we will not go over these possible derivations in this paper, but they are included in the appendices; the resulting predictions are the same across the board.

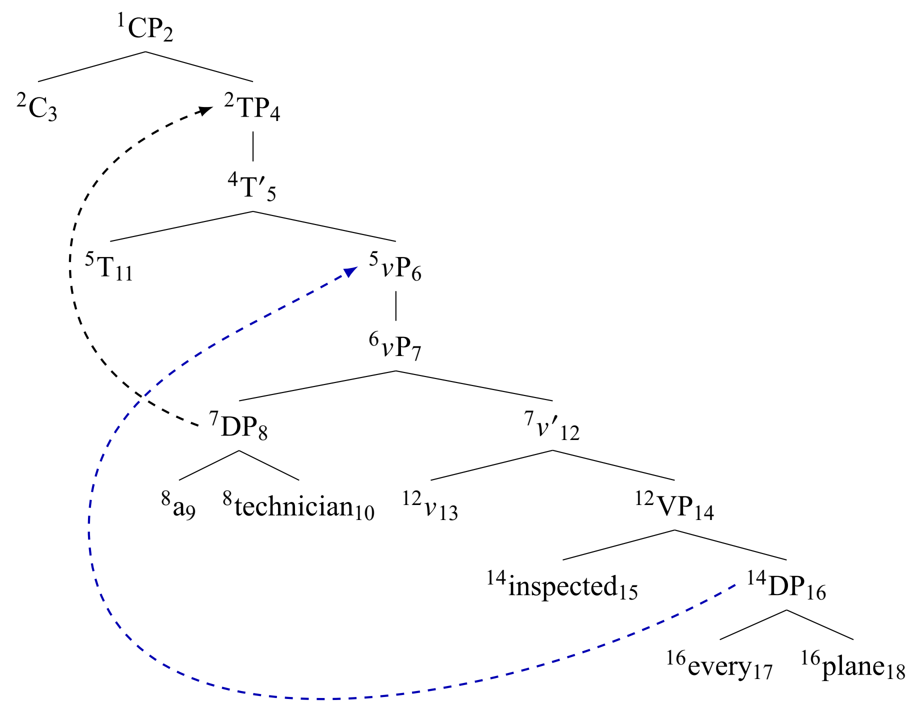


Figure 5: Annotated derivation tree for surface scope reading of *A technician inspected every plane*.

5.3 Example 2: Subject and Object vs. Negation

Next we will discuss the relative scope configurations of subject and object universal quantifiers and sentential negation, as tested by Lee (2009). We will only offer an explicit analysis of the English case due to the aforementioned complexities involved in a full analysis of Korean. However, at the end of this subsection we will hint at how the SLD Principle could cover the Korean case as well.

As discussed in §2, Lee (2009) provides evidence suggesting that in negated sentences with *every* in subject position, as in (31a) (repeated below), there is a preference for surface scope, i.e., for *every* to scope above negation. That is, the preferred reading is that no kid fed the doves, rather than the weaker reading that at least one kid withheld their food.

(31a) According to the story, every kid didn't feed the doves in the park. (Lee 2009, p. 93)

We also saw that this was problematic for Wurmbrand's theory of scope processing, as in this case the preferred reading was one in which the subject was interpreted farther away from its merge position, meaning that there was more movement at LF. In showing that the SLD Principle makes the right predictions, we will use the simpler (69):

(69) Every student did not pass the test.

Consider Figure 6, which is the annotated derivation tree for the inverse scope interpretation of (69). Here the subject's movement is PF-only because the licensee feature for the subject's movement to spec-TP is `-nom`$_{\mathrm{P}}$, rather than `-nom`. As a result, the derived PF is as expected, and the LF representation is as in (70), generating the weak *not* > *every* interpretation.

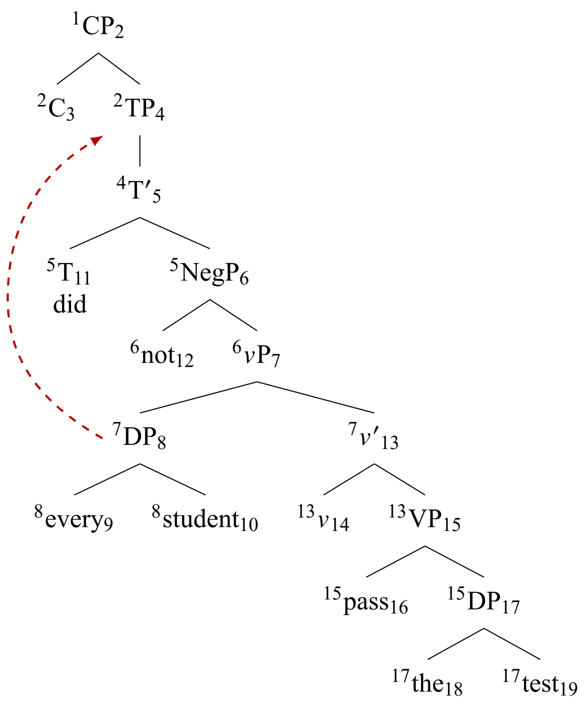


Figure 6: Annotated derivation tree for *Every student did not pass the test* (inverse scope)

(70) [C [T [NEG [[EVERY STUDENT] [V [PASS [THE TEST]]]]]]]

With respect to the SLD of this parse, there is only one mover here, so *every student* is the only constituent with a defined location differential. The existence of *every student* is first predicted when it is unmoved at step 4. At this point, *every student* is assigned a PF address (that of spec-TP), but not an LF address, since knowing that *every student* PF-moved to spec-TP does not tell us where it takes scope. It only receives an LF address when it is unmerged at step 7 (the outdex of the immediately dominating **merge** node, labeled *v*P), whereupon it is established that the subject takes scope in spec-*v*P. Therefore, the location differential of *every student*—and hence the SLD of the parse as a whole—is $7 - 4 = 3$.

As for the surface scope interpretation of this sentence, we generate this by replacing the PF-only licensee feature $\texttt{-nom}_{\texttt{P}}$ with the PF+LF licensee feature $\texttt{-nom}$. Upon making this switch, the final PF representation is exactly the same as it was in the inverse scope derivation, but at LF the subject scopes in spec-TP, leaving us with the surface-scope structure in (71):

(71) [C [[EVERY STUDENT] [$\lambda_{\text{in}(\texttt{-nom})}$ [T [NEG [$t_{\text{in}(\texttt{-nom})}$ [V [PASS [THE TEST]]]]]]]]]

Since the PF structures of the surface- and inverse-scope derivations are identical, the path the parser takes through the derivation tree will be identical to that of the inverse-scope derivation, meaning that as far as the annotated derivation tree is concerned, the only difference between surface scope and inverse scope is that the red PF-only movement arrow is replaced with a black PF+LF-movement arrow. However, this single difference in the derivation leads to an important parallel distinction in the parse and the SLD thereof. This time, when the subject is unmoved in step 4, rather than knowing only its PF position, we are now immediately aware of both the subject's PF address and its LF address: namely, spec-TP. There are no steps at which *every student* is predicted but missing a PF or LF address, meaning that the SLD for this parse is 0. Thus, according to the SLD Principle, since the surface scope parse has a lesser SLD than the inverse scope parse, the former is rightly predicted to be less costly than the latter.

Next we turn to the relative scope preferences for objects and negation. Recall that what Lee (2009) found was that in English, there was a strong preference for universally-quantified direct objects—such as *every candle* in (32a), repeated below—to scope under negation (*not* > *every*), in contrast to subjects.

(32a) According to the story, Cindy didn't light every candle last night. (Lee 2009, p. 124)

Much like we did with subjects, we will use the structurally simpler (72) as our test case:

(72) Mary did not feed every patient.

Starting with inverse scope (*every* > *not*), the annotated derivation tree for the parse of (72) on its inverse scope reading can be seen in Figure 7. The PF tree arrived at by this derivation is as in (73a) (with non-"trace" ε replaced by relevant head names for readability's sake), and the LF tree is as in (73b) (broken up into two lines), with *every patient* outscoping negation.

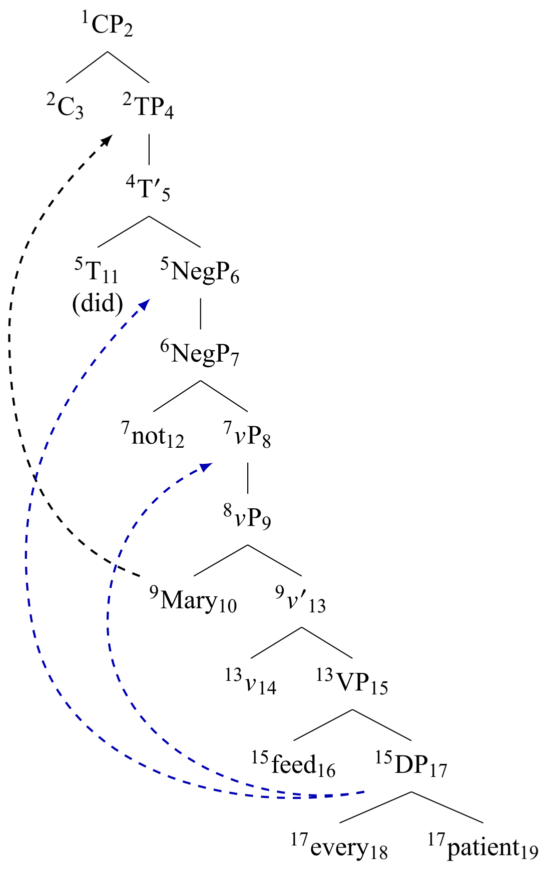


Figure 7: Annotated derivation tree for *Mary did not feed every patient* (inverse scope)

(73) a. [C [Mary [T(=did) [not [ε [v [feed [every patient]]]]]]]]

b. [C [MARY [$\lambda_{\text{in}(\texttt{-nom})}$ [T [[EVERY PATIENT] [$\lambda_{\text{in}(\texttt{-sc}_\text{L})}$ [NEG [$t_{\text{in}(\texttt{-sc}_\text{L})}$ [$\lambda_{\text{in}(\texttt{-sc}_\text{L})}$ [$t_{\text{in}(\texttt{-nom})}$ [V [FEED $t_{\text{in}(\texttt{-sc}_\text{L})}$]]]]]]]]]]]

A few things are worth noting about the LF tree in (73b). The first is that while *Mary* takes scope in spec-TP, because it is a proper name there is no semantic difference between it scoping there and it scoping in its merge position of spec-*v*P (at least on most analyses of proper names). We have opted for the former in order to obtain the lowest possible SLD for this parse, since we want to compare the least costly parse for each of inverse and surface scope. The second observation is that *every patient* undergoes QR twice: once to the edge of *v*P, and once above negation. The same licensee feature ($\texttt{-sc}_\text{L}$) is used both times, meaning that *every* has two $\texttt{-sc}_\text{L}$ features, one that is checked by a `+sc` feature in the *v* head, and one that is checked by a `+sc` feature in the negation head. After each move there is lambda abstraction ($\lambda_{\text{in}(\texttt{-sc}_\text{L})}$). After first lambda abstracting over the trace left by the first QR of *every patient*, this predicate is immediately saturated by the higher trace left by the second QR of *every patient*, meaning that we are left with the same result as if there had been no lambda abstraction at all. The second lambda abstraction (just above negation) is the final one, creating the predicate fed to *every patient*. The semantic result is the same as if *every patient* had moved above negation in one straight shot. Breaking the QR down into two steps is adopted for syntactic reasons only, based on the assumption that *v*P is a movement domain (cf. the discussion in §2); abandoning this assumption does not meaningfully alter the results.

Next, we compute the SLD for this parse. For *Mary*, the location differential is zero: because its one movement is a PF+LF movement, once this movement is undone *Mary*'s PF and LF addresses (both spec-TP) are immediately identifiable. As for *every patient*, its existence is first predicted when its final LF-only movement is undone in step 6; at this point, its LF position (just above negation) is known, but its PF location is not. When the other LF-only movement is undone at step 8, *every patient*'s PF location is still unknown, since we still have not undone a PF-relevant operation. Instead, *every patient*'s PF location is not known until it is unmerged from the VP in step 15. Thus, *every patient*'s location differential—and therefore the SLD of the parse as a whole—is $15 - 6 = 9$.

Turning next to surface scope (*not* > *every*), this is obtained by getting rid of *every*'s second $\texttt{-sc}_\text{L}$ feature, as well as *not*'s `+sc` feature. When this is done, *every patient*'s type-necessitated LF-only movement to spec-*v*P still takes place, but this time the DP stays there, remaining below negation. Thus, while the PF representation of the sentence remains the same, the LF representation looks as in (74):

(74) [C [MARY [$\lambda_{\text{in}(\texttt{-nom})}$ [T [NEG [[EVERY PATIENT] [$\lambda_{\text{in}(\texttt{-sc}_\text{L})}$ [$t_{\text{in}(\texttt{-nom})}$ [V [FEED $t_{\text{in}(\texttt{-sc}_\text{L})}$]]]]]]]]]

The annotated derivation tree for the parse can be seen in Figure 8. As before, the location differential of *Mary* is 0, as its PF and LF addresses are both predicted when **unmove** takes place at step 4. For *every patient*, the location differential is 7, since its existence is predicted at step 7 and it is assigned a PF address at step 14. The SLD Principle again derives the right prediction: the SLD for the costlier inverse scope (9) is greater than the SLD for the less costly surface scope (7).

With respect to Korean, where speakers prefer for both subjects and objects to scope over negation, as discussed in §2 any adequate syntactic analysis will have to account for the fact that in Korean, subjects and direct objects linearly precede the verb and negation at PF. On an antisymmetric view of syntax (Kayne 1994), this means that the two DPs (or constituents containing them) must c-command the verb and negation. But if the subject and direct object can each scope at their PF

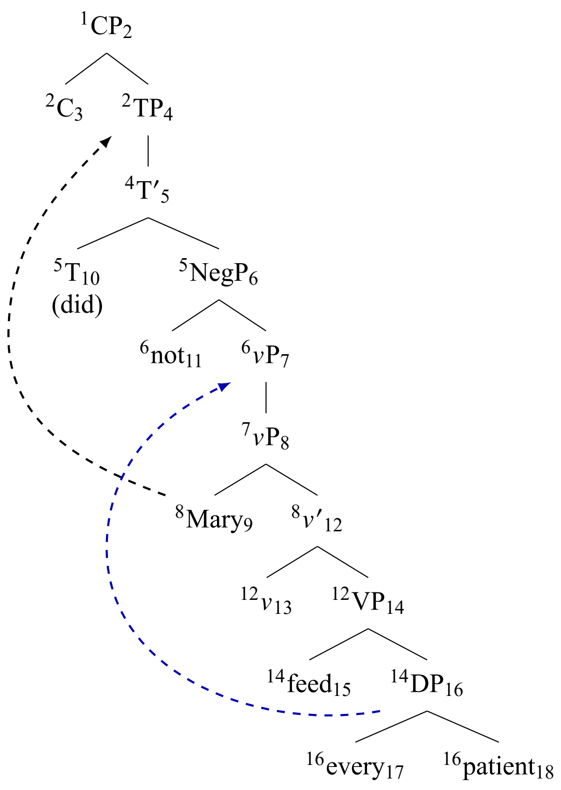


Figure 8: Annotated derivation tree for *Mary did not feed every patient* (surface scope)

positions, then in both cases there is an *every* > *not* derivation in which there are no LF-only or PF-only movements, and any and all movements are PF+LF movements. In this case, the parse's SLD will be zero, since both PF and LF locations are known as soon as a PF+LF movement is undone. Meanwhile, in order to derive a *not* > *every* reading, one of two things will have to happen: the subject/object's movement from below to above negation could be a PF-only movement—meaning it would scope below its post-**move** position and below negation—or the negation (or some constituent containing it) could undergo some additional LF-only movement to a position higher than the subject/object. But both of these possibilities result in a non-zero SLD, since each requires either a PF-only or LF-only movement. We would therefore predict inverse scope (*not* > *every*) to incur a greater cost than surface scope (*every* > *not*) for both subjects and objects, as desired.

In summary, the SLD Principle successfully accounts for the English scope preferences for subject over negation and negation over direct object, and there are promising signs that it could be extended to account for preferences in Korean, though this will no doubt depend on one's syntactic analysis of Korean.

5.4 Example 3: Cyclic QR

Next, we will account for the cyclic QR observations analyzed by Wurmbrand (2018). There are two types of observation that we wish to account for. The first are the within-sentence observations: for each of the sentences in (8), surface scope is easier to process than inverse scope. The second are the across-sentence observations: an inverse-scope interpretation of (8a) is easier to process than an inverse-scope interpretation of (8b), which in turn is easier than an inverse-scope interpretation of (8c).

(8) a. A technician inspected every plane.
b. A technician tried to inspect every plane.
c. A technician decided to inspect every plane.

Recall that Wurmbrand accounts for these facts by means of an analysis in which clausal complements vary in their size, with *try*-complements being *v*Ps, and *decide*-complements including an additional future-shifting head WOLL. Because *v*P is a movement domain, *every plane* must undergo QR twice to generate inverse scope in (8b) (first above the embedded *v*P, then above the matrix *v*P), while in (8c) a third iteration of QR is required, since *every plane* must also make a stop in spec-WOLLP. Since only one iteration of QR is required for (8a), on Wurmbrand's analysis we generate the correct across-sentence predictions, in addition to the within-sentence ones.

To illustrate the efficacy of the SLD Principle, we will adopt precisely the same syntactic analysis as Wurmbrand, leading to the same predictions with respect to processing difficulty. However, as will be discussed later, we do not need every feature of her syntactic analysis to capture the scope processing facts: the SLD Principle is flexible enough that even with important changes to the syntactic analysis, the correct results are still derived.

In order to determine whether the SLD principle garners the right results, we first need to find the SLDs of the surface scope and inverse scope interpretations for each sentence in (8). In §5.2 we already did this for (8a); our results were SLDs of 8 for surface scope and 11 for inverse scope, correctly predicting a preference for surface over inverse scope. Next we move on to (8b). As per Wurmbrand's theory, *try* takes a *v*P complement. Given our aforementioned assumption that all objects must undergo QR because of a type mismatch, on a surface scope interpretation *every plane* will QR above the embedded *v*P and just below *try*. As for *a technician*, whether it scopes in its merge or final position does not matter, since either way it will scope above *every plane*. However, we will have it scope in spec-TP in order to minimize the SLD—again, we evaluate each reading on its least costly parse. The resulting annotated derivation tree can be seen in Figure 9; the ensuing PF tree is as one would expect, and the LF tree is as in (75):[26]

(75) [C [[A TECHNICIAN] [$\lambda_{\text{in}(\texttt{-nom})}$ [T [$t_{\text{in}(\texttt{-nom})}$ [V [TRY
[[EVERY PLANE] [$\lambda_{\text{in}(\texttt{-sc}_{\text{L}})}$ [PRO [V [INSPECT [$t_{\text{in}(\texttt{-sc}_{\text{L}})}$]]]]]]]]]]]]

In calculating the SLD for this parse, we must look at the two movers: *a technician* and *every plane*. For the former, the location differential is zero because the movement is PF+LF movement, meaning the final resting spot at PF and LF can be determined as soon as this movement is undone at step 4. As for *every plane*, this DP is assigned an LF location at step 15 when it is unmoved, but is not assigned a PF address until step 20, when it is unmerged. Hence, the location differential for *every plane*, and the SLD for the whole parse, is $20 - 15 = 5$.

Next up is the inverse scope interpretation of (8b). In order to get this reading, just like for Wurmbrand's analysis *every plane* must undergo QR twice, once above the embedded *v*P and once above the matrix *v*P. In addition, *a technician*'s movement to spec-TP must be PF-only, meaning that it scopes in its merge position just below *every plane*. The annotated derivation tree can be seen in Figure 10; the resulting PF tree is the same as before, and the LF tree is as in (76):

[26] We assume, contra Wurmbrand (2001), that the complements of *try*-type verbs include PRO. Eliminating this assumption does not change our results.

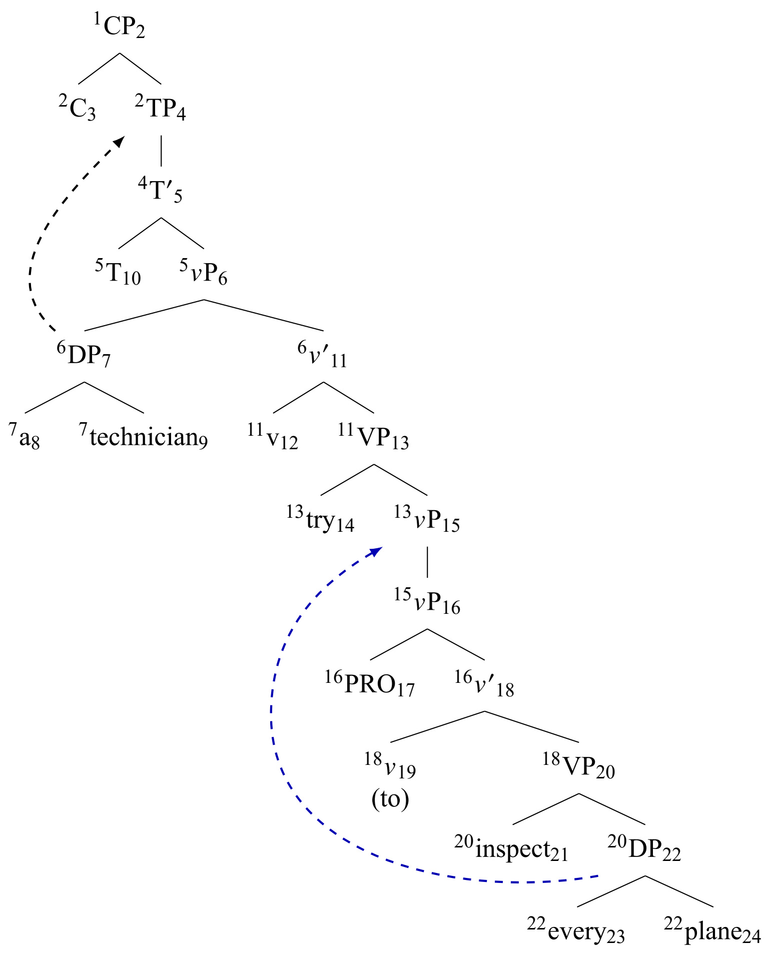


Figure 9: Annotated derivation tree, *A technician tried to inspect every plane* (surface scope)

(76) [C [T [[EVERY PLANE] [$\lambda_{\text{in(-sc}_\text{L})}$ [[A TECHNICIAN] [V [TRY [$t_{\text{in(-sc}_\text{L})}$ [$\lambda_{\text{in(-sc}_\text{L})}$ [PRO [V [INSPECT $t_{\text{in(-sc}_\text{L})}$]]]]]]]]]]]

When calculating the SLD for this parse, we again have two movers to account for. This time, the matrix subject has a non-zero location differential, since its existence is predicted when it is unmoved in step 4, but it is not assigned an LF location until it is unmerged in step 7, leading to a location differential of $7 - 4 = 3$. As for *every plane*, its existence is first predicted at step 6, when it is unmoved. At this point, it has a definite LF location, but its PF location is yet to be determined. Undoing the other LF-only movement (above the embedded *v*P) does not change this fact, so *every plane* is not assigned a PF address until it is unmerged from the complement of *inspect* at step 21. Thus, *every plane*'s location differential is $21 - 6 = 15$, leading to an SLD of $3 + 15 = 18$. This generates the right predictions both within-sentence—we predict surface scope to be easier than inverse scope, since $5 < 18$—and across-sentence, since inverse scope for (8a) is predicted to be easier than inverse scope for (8b) ($11 < 18$).

Finally, we have (8c), the example with *decide* + infinitive. We will not include the annotated derivation tree for the surface scope parse in the body of this paper, as it looks minimally different from the surface scope derivation tree for (8b): the only difference is the inclusion of WOLL (and

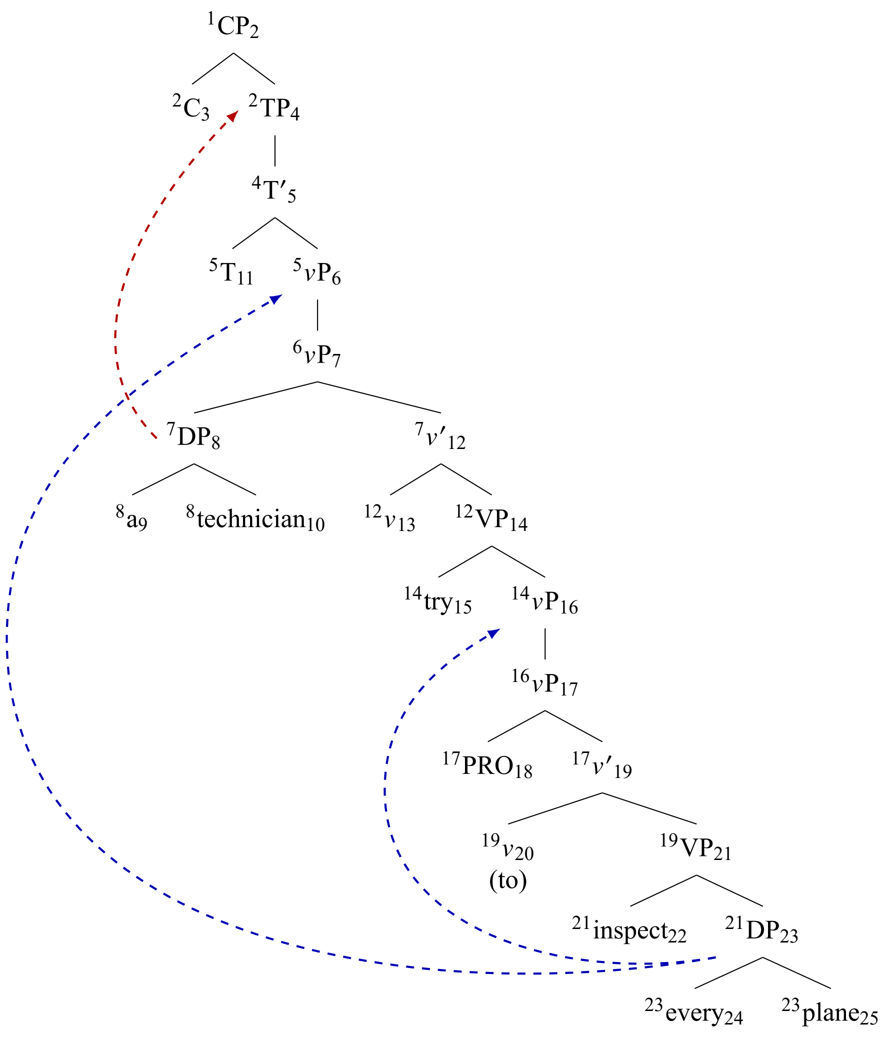


Figure 10: Annotated derivation tree, *A technician tried to inspect every plane* (inverse scope)

WOLLP), which has no effect on the location differentials for either the subject or the embedded object. As a result, we again get 5 as the SLD for the surface scope parse of (8c), since the subject has a location differential of 0—it scopes where it sits—and the object has a location differential of 5, due to the five steps between when it is unmoved and when it is assigned a PF address.

This leaves only the inverse scope interpretation of (8c). As per Wurmbrand's theory, *every technician* has to undergo three iterations of QR this time: once directly above the embedded *v*P, once above WOLLP, and once above the matrix *v*P. The annotated derivation tree for this parse can be seen in Figure 11; the derived LF structure is as in (77):

(77) [C [T [EVERY PLANE] [$\lambda_{\text{in(-sc}_\text{L})}$ [[A TECHNICIAN] [V [DECIDE [$t_{\text{in(-sc}_\text{L})}$ [$\lambda_{\text{in(-sc}_\text{L})}$ [WOLL [$t_{\text{in(-sc}_\text{L})}$ [$\lambda_{\text{in(-sc}_\text{L})}$ [PRO [V [INSPECT $t_{\text{in(-sc}_\text{L})}$]]]]]]]]]]]]]

Finally, we must determine the SLD for this parse. As before, the subject has a location differential of 3: its existence is predicted in step 4, and it is not assigned an LF location until it is unmerged in step 7. As for *every plane*, its existence is again predicted upon being unmoved at

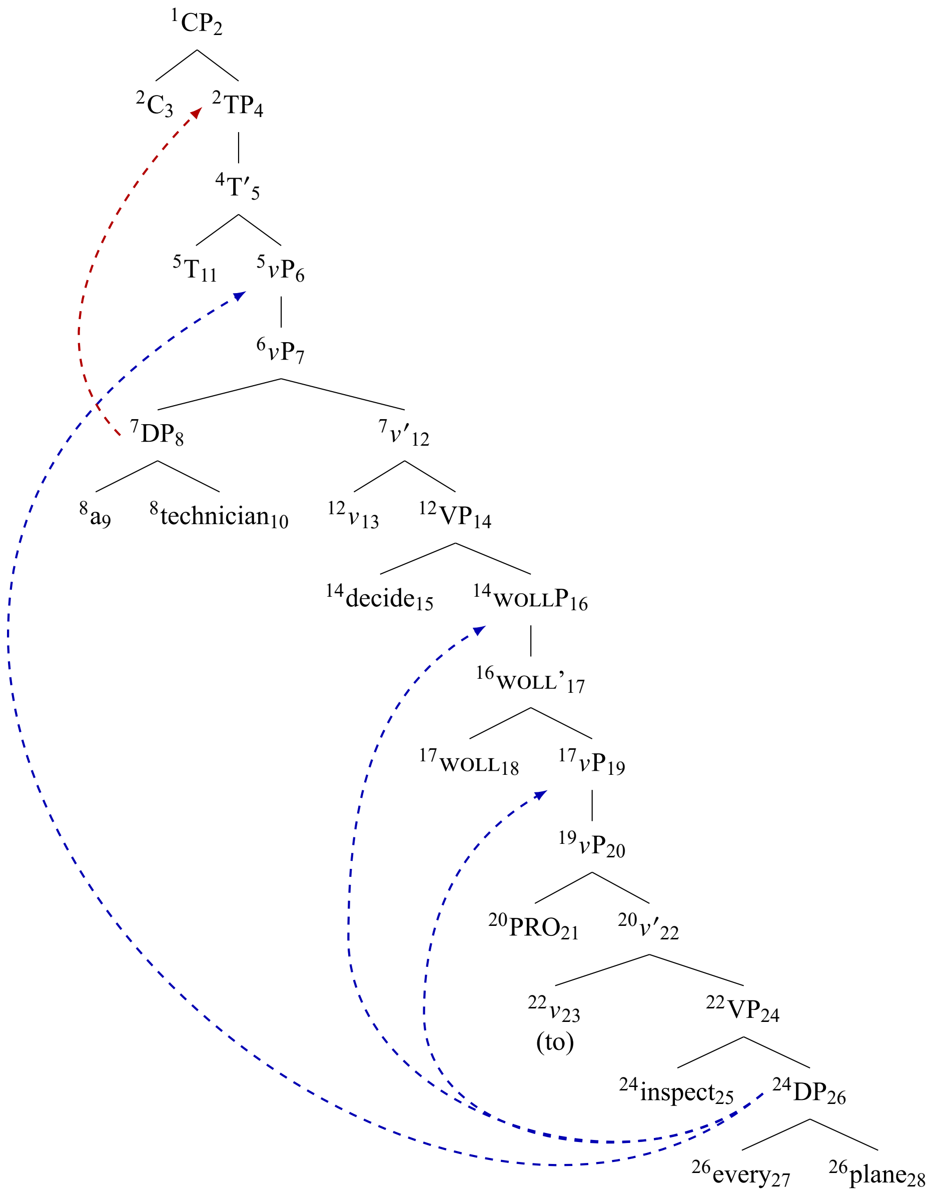


Figure 11: Annotated derivation tree, *A technician decided to inspect every plane* (inverse scope)

step 6, at which point it has an LF address and no PF address. Moreover, neither of the other two unmoves suffices to determine the PF address of *every plane*, so it is not assigned a PF address until it is unmerged at step 24. Thus, *every plane*'s location differential is $24 - 6 = 18$, leading to an SLD of $3 + 18 = 21$ for the parse. This again gives us the correct within-sentence prediction, since surface scope is easier than inverse scope ($5 < 21$), as well as the right across-sentence predictions, since inverse scope for (8a) is easier than for (8b), which is easier than for (8c) ($11 < 18 < 21$).

We have thus derived all of the desirable within- and across-sentence predictions, just like Wurmbrand (2018). A table with all of the SLDs for the three sentences on both surface and inverse scope interpretations can be seen in Figure 12. But there is one blip in this table that is worth discussing: we generate the seemingly odd prediction that surface scope for *try* and *decide* sentences is easier to process than for the monoclausal case ($5 < 8$). This strange prediction—which, while to our knowledge is untested, nonetheless seems *prima facie* false—owes its existence to two distinctions between the parses.

	Surface Scope	**Inverse Scope**
monoclausal	8	11
***try* + inf**	5	18
***decide* + inf**	5	21

Figure 12: SLDs for surface and inverse scope parses for the sentences in (8)

The first is that in the monoclausal case, the subject *a technician* is unmerged from *v*P and then broken up and scanned after *every plane* is predicted but before it is assigned a PF address, meaning that the subject adds four steps to the object's location differential (**unmerge** from *v*P, **unmerge** of the subject into D and N, and the **scan** of *a* and *technician*). Meanwhile, for *try* and *decide*, on a surface scope interpretation the matrix subject *a technician* is fully taken care of before the existence of *every plane* is predicted, meaning that it does not add to the latter's location differential. However, the embedded subject PRO does add some steps, since it is unmerged and scanned after *every plane* is predicted and before it is assigned a PF address. But since PRO is a single lexical item, rather than a phrase made up of multiple lexical items, the unmerging and scanning of PRO takes only two steps, rather than the four that *a technician* takes in the monoclausal case. Thus, the fact that PRO can be immediately scanned upon being unmerged subtracts two steps from *every plane*'s location differential for the *try* and *decide* sentences.

The second distinction between the parses comes from the scanning of the (matrix) tense node. Because lexical items are scanned in linear order from left to right, T must be scanned immediately after the (matrix) subject, since it immediately follows it. Looking at the monoclausal case, we know from the prior discussion that the subject is scanned after the object is predicted and before it is assigned a PF address. Since the next step is the scanning of T, this scan operation similarly intervenes, adding one to the location differential of *every plane*. Meanwhile, in the case of *try* and *decide*, T is scanned before the existence of *every plane* is predicted, so it does not intervene. This leads to a one-step reduction in *every plane*'s location differential, which in conjunction with the two steps saved thanks to PRO leads to the observed three-step distinction.

Thus, if the SLD Principle is to be maintained, these two distinctions must be either eliminated or neutralized in their impact. Fortunately, this can be done in one fell swoop, if we eliminate the so-far-adopted assumption that object quantificational DPs must undergo QR for reasons of type mismatch. After all, this type mismatch is an artifact of a particular compositional semantics, and there are plenty of alternative theories of compositionality according to which quantificational DPs can scope in their merge position just fine.[27] If we eliminate this compositional stipulation, then for all three sentences in (8) there exists a surface-scope interpretation whose parse has an SLD of zero, since each quantifier scoping in its PF position leads to a surface scope interpretation. Note that this will also not reverse any of our previous results, as it will merely increase the difference in SLDs between inverse- and surface-scope interpretations: surface scope interpretations will always include derivations whose parses similarly have SLDs of zero.

Before moving on to the predicted difference between cyclic QR and cyclic *wh*-movement, one important thing is worth noting about the extent to which the success of the SLD Principle hinges on the particulars of Wurmbrand's syntactic analysis. Because of her emphasis on the number of QR operations, for Wurmbrand it matters a great deal what—and more importantly, how many—QR operations there are before a quantificational DP reaches its final LF landing site. For example, if it turns out that WOLLP is not a movement domain, then *decide*-type infinitives can generate inverse scope readings in essentially the same way as for *try*-type infinitives, with one movement to the edge of the embedded *v*P, and one to the edge of the matrix *v*P. In this case, Wurmbrand would no longer predict a difference in processing difficulty.

However, this is not the case for the SLD Principle. Consider the three-step difference in SLD between (8b) and (8c) on their inverse scope interpretations. Given that most of the steps in these two parses are the same, we can reasonably ask which parse steps in the parse of (8c) in Figure 11 are responsible for this three-step difference. Not surprisingly, these steps are the ones that involve WOLL or WOLLP: (I) the undoing of *every plane*'s movement to spec-WOLLP in step 16, (II) the unmerging of *v*P from WOLL in step 17, and (III) the scanning of WOLL in step 18. Now suppose that we say that WOLLP is not a movement domain, and *every plane* can LF-move straight from the lower *v*P to the higher *v*P. This would eliminate the need for the first of these parse steps (since there would be no movement to spec-WOLLP that would need undoing), reducing the SLD for the parse as a whole. However, this would have no impact on the other two parse steps, since WOLL still needs to be unmerged and scanned. In other words, the difference in SLD between (8b) and (8c) would shrink from three to two, but we would nonetheless predict the latter to be harder to process than the former on an inverse scope interpretation. In fact, even if we were to remove all movement domains whatsoever and suppose that in all cases, *every plane* moves straight from its merge position to its final landing spot in one go, we would still predict a difference in processing difficulty, since eliminating these extra movements would have no impact on the additional structure that has to be unmerged and scanned when going from (8a) to (8b), and from (8b) to (8c). This illustrates another advantage to the SLD Principle: it can survive a variety of small (and not-so-small) changes on the syntactic end of things, as long as certain basic assumptions are maintained.

[27] While many theories allowing *in situ* DP interpretation eschew QR altogether, achieving scope reconfiguration by other means, this is obviously not an in-and-of-itself necessary condition on theories permitting *in situ* interpretation. See for example Keenan's (2005) use of "rich" (essentially, highly type-flexible) DP denotations, where all DPs can be interpreted *in situ*, but scope rearrangements are still obtained via QR.

5.5 Example 4: *Wh*-movement vs. Cyclic QR

Finally, let us look at the case of overt *wh*-movement. As discussed in §2, this is problematic for Wurmbrand's analysis as it is originally proposed, since at least intuitively (overt) cyclic *wh*-movement is considerably easier than (covert) cyclic QR. Or, more concretely, (30a) seems easier to process than an inverse scope interpretation of (30b):

(30) Wurmbrand 2018, p. 25 (based on her (29)):

   a. What did a technician say that John inspected?

   b. A technician said that John inspected every plane.

As a result, Wurmbrand is forced to seek modifications to her proposal that would account for this apparent distinction, modifications that have their own, altogether different problems.

However, the SLD Principle gets the (*prima facie*) correct results here without any further stipulation, at least if we follow Karttunen (1977) in treating moved *wh*- phrases as "scoping where they sit" in spec-CP. To show that this is the case, we will use (78) as our test example:

(78) What did a technician try to inspect?

The annotated derivation tree for (78) can be seen in Figure 13. (We assume the subject *a technician* scopes in spec-TP instead of its merge position, adopting the aforementioned principle that what is relevant is the *least* costly parse for each interpretation.) Since all of the movements in this derivation are PF+LF movements, and since as soon as a PF+LF movement is undone the PF and LF addresses of the moved constituent are immediately known, the SLD for this parse is zero. We therefore rightly predict that overt scope-taking operations are easier to process than their covert counterparts.

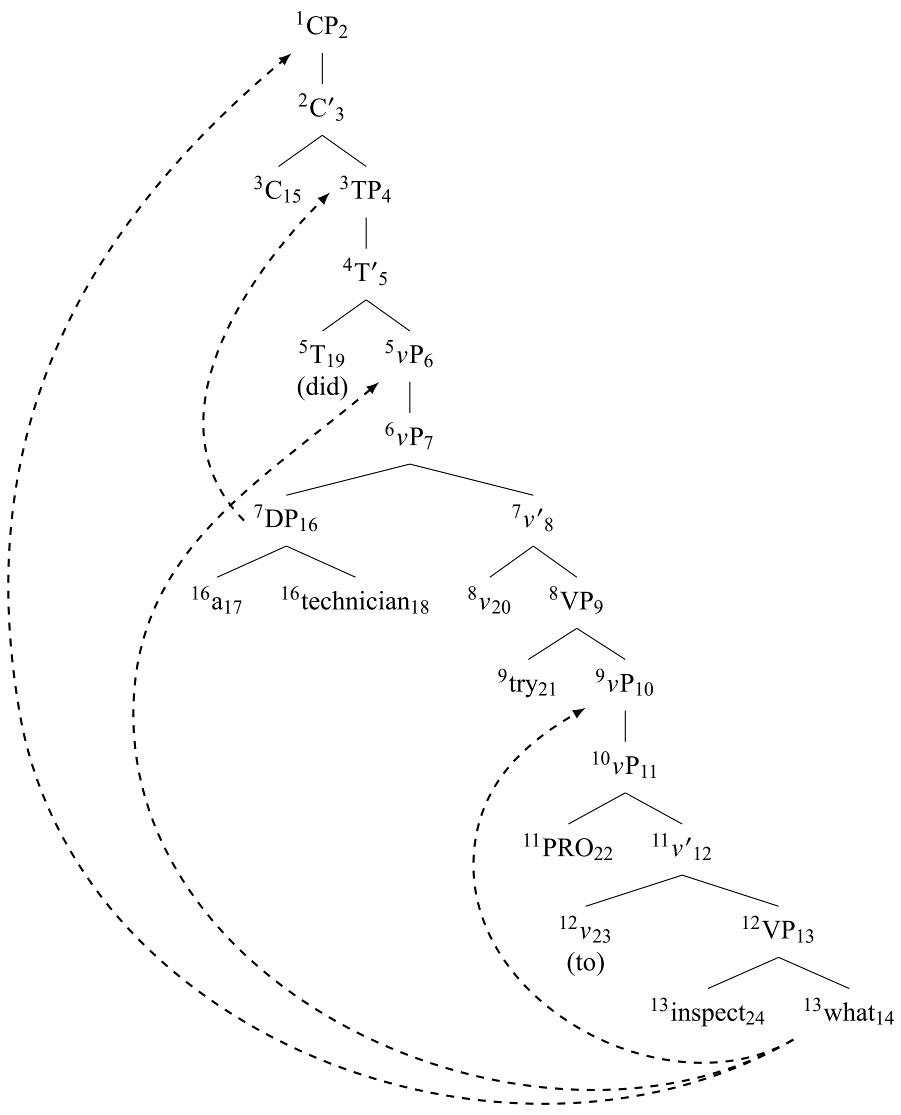


Figure 13: Annotated derivation tree, *What did a technician try to inspect?*

An interesting follow-up question—and one that we must unfortunately leave for future work—is what happens when we turn from overt *wh*-movement to cases in which some or all *wh*- phrases stay *in situ* at PF. This includes languages like Japanese, where true overt *wh*-movement does not exist, as well as English, where only one *wh*- phrase undergoes overt *wh*-movement to the left edge of a given clause, with the others appearing *in situ*:

(79) a. Which technician inspected which plane?
b. * Which technician [which plane]$_1$ inspected t_1?

There are two reasons why we cannot yet offer an account of these cases. The first is that to our knowledge, the empirical facts on the processing of moved vs. (PF-)*in situ wh*- phrases have yet to be clearly established. The second is that there is also not a settled account in the theoretical literature of what happens to PF-*in situ wh-* phrases at LF. According to the classic analysis of Huang (1982), PF-*in situ wh-* phrases essentially do at LF what moved *wh-* phrases do at both PF and LF: namely, they move to a clause-initial position. In this case, for *wh-in situ* languages we would expect a sentence like (78) to be at least as difficult to process as (8b)—*modulo* any intervening principles that might ease processing—since the *wh-* phrases would have to undergo QR-like LF-only movement. However, Kotek (2016) argues for a view in which PF-*in situ wh-* phrases needn't LF-move all the way to spec-CP, and only LF-move as far as is needed for interpretive reasons—perhaps not at all in many cases.[28] In this case, the SLD Principle would not predict the existence of PF-*in situ wh-* phrases to have a significant impact on processing difficulty, except when LF movement is required to prevent uninterpretability, e.g., in the case of intervention effects. Thus, in order to determine whether the SLD Principle generates the right results with respect to the processing of *in situ wh-* phrases, more work needs to be done to establish what the actual results are, as well as what the derived PF and LF structures for the relevant sentences are.

Finally, it is worth noting that the fact that cyclic *wh*-movement adds nothing to the SLD of a parse does not mean that we necessarily predict that all (grammatical) *wh*-movement should be an absolute breeze to process. Put simply, the SLD Principle is not the only factor that determines processing difficulty for a given sentence, and other factors may make certain instantiations of *wh*-movement quite difficult to process. The important thing for our purposes is that whatever makes such *wh*-movements difficult to process is different from whatever makes inverse scope of (30b) difficult to process, hence the apparent difference between (30b) and (30a).

6 Conclusion

In this paper we have discussed evidence, much of which was previously discussed in detail by Wurmbrand (2018), suggesting that QR is not a clause-bounded operation, contrary to prior observations reported in the theoretical literature. Instead we have followed Wurmbrand in adopting the view that extraclausal QR, though fully grammatical, is nonetheless difficult to process—often prohibitively so. We have additionally followed Wurmbrand in postulating that the processing difficulty of extraclausal QR depends on the size of the embedded clause, with the complements of *try*-type verbs being more conducive to extraclausal QR than the complements of *decide*-type verbs.

While we have argued against the particular theory of scope processing difficulty that Wurmbrand offers to account for these observations, we have proposed an alternative metric that is in keeping with a proposed revision that she suggests, in which processing difficulty is in part dependent on the severity of the mismatch between PF and LF representations. This theory was couched in a top-down parser for Minimalist Grammars, thereby embedding it within a framework that has already been used to successfully account for a variety of observations on syntactic processing from the experimental literature. The metric was then shown to make the right predictions for all of the data discussed by Wurmbrand, as well as those that were problematic for her original account.

By using a top-down MG parser to formulate our analysis of scope processing difficulty, we have added to a growing body of work dedicated to using such parsers to account for a variety of syntactic processing effects. However, the SLD Principle looks quite different from the processing metrics that have been developed in the prior literature, such as those based on the notion of tenure

[28]For experimental evidence in favor of this view, see Kotek & Hackl 2013.

discussed in §4. We leave for future work the issue of whether—and if so, how—to integrate these processing metrics into a more cohesive and unified picture of syntactic-semantic processing.

Of more pressing and direct concern for the theory as it currently stands is the need for more experimental work on scope processing. Any theory of scope processing difficulty is inherently constrained by the set of empirical facts available to be accounted for. Our hope is that the introduction of a robust and thus far successful scope processing metric will encourage more empirical work testing the predictions of the SLD Principle, so that it may be either further refined or replaced with an equally predictive and more empirically adequate alternative.

References

Anderson, Catherine. 2004. *The structure and real-time comprehension of quantifier scope ambiguity*. Evanston, IL: Northwestern University PhD dissertation.

Cecchetto, Carlo. 2004. Explaining the locality conditions of QR: Consequences for the theory of phases. *Natural Language Semantics* 12(4). 345–397.

Chomsky, Noam. 1965. *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Chomsky, Noam & George A. Miller. 1963. Introduction to the formal analysis of natural languages. In R. Duncan Luce, Robert R. Bush & Eugene Galanter (eds.), *Handbook of Mathematical Psychology, Vol. II*, 269–321. New York, NY: Wiley.

De Santo, Aniello. 2019. Testing a Minimalist grammar parser on Italian relative clause asymmetries. In *Proceedings of the ACL Workshop on Cognitive Modeling and Computational Linguistics (CMCL) 2019*, June 6 2019, Minneapolis, Minnesota.

Farkas, Donka & Anastasia Giannakidou. 1996. How clause-bounded is the scope of universals? In Teresa Galloway & Justin Spence (eds.), *Semantics and Linguistic Theory (SALT)* 6, 35–52.

Fodor, Janet Dean & Ivan Sag. 1982. Referential and quantificational indefinites. *Linguistics and Philosophy* 5(3). 355–398.

Fox, Danny. 2000. *Economy and Semantic Interpretation*. Cambridge, MA: MIT Press.

Fox, Danny. 2002. Antecedent-contained deletion and the copy theory of movement. *Linguistic Inquiry* 33(1). 63–96.

Fox, Danny. 2003. On logical form. In Randall Hendrick (ed.), *Minimalist Syntax*, 82–123. Oxford: Blackwell Publishers.

Gerth, Sabrina. 2015. *Memory limits in sentence comprehension: A structural-based complexity metric of processing difficulty*. Potsdam: Universität Potsdam PhD dissertation.

Graf, Thomas, Brigitta Fodor, James Monette, Gianpaul Rachiele, Aunika Warren & Chong Zhang. 2015. A refined notion of memory usage for Minimalist parsing. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, 1–14. Chicago, IL: Association for Computational Linguistics.

Graf, Thomas, James Monette & Chong Zhang. 2017. Relative clauses as a benchmark for Minimalist parsing. *Journal of Language Modelling* 5(1). 57–106.

Hackl, Martin, Jorie Koster-Hale & Jason Varvoutis. 2012. Quantification and ACD: Evidence from real-time sentence processing. *Journal of Semantics* 29(2). 145–206.

Hankamer, Jorge & Ivan Sag. 1976. Deep and surface anaphora. *Linguistic Inquiry* 7(3). 391–426.

Heim, Irene & Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Oxford: Blackwell.

Hornstein, Norbert. 1994. An argument for minimalism: The case of antecedent-contained deletion. *Linguistic Inquiry* 25(3). 455–480.

Huang, C.T. James. 1982. Move WH in a language without WH movement. *The Linguistic Review* 1(4). 369–416.

Joshi, Aravind K. 1990. Processing crossed and nested dependencies: an automaton perspective on the psycholinguistic results. *Language and Cognitive Processes* 5(1). 1–27.

Karttunen, Lauri. 1977. Syntax and semantics of questions. *Linguistics and Philosophy* 1(1). 3–44.

Kayne, Richard. 1994. *The Antisymmetry of Syntax*. Cambridge, MA: MIT Press.

Keenan, Edward. 2005. *In situ* interpretation without type mismatches. UCLA, Ms.

Kennedy, Christopher. 1997. Antecedent-contained deletion and the syntax of quantification. *Linguistic Inquiry* 28(4). 662–688.

Kobele, Gregory M., Sabrina Gerth & John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In Glyn Morrill & Mark-Jan Nederhof (eds.), *Formal Grammar: 17th and 18th International Conferences*, 32–51.

Kotek, Hadas. 2016. Covert partial *wh*-movement and the nature of derivations. *Glossa: a journal of general linguistics* 1(1): 25. 1–19.

Kotek, Hadas & Martin Hackl. 2013. An experimental investigation of interrogative syntax/semantics. In Maria Aloni, Michael Franke & Floris Roelofson (eds.), *Proceedings of the 19th Amsterdam Colloquium*, 147–154.

Kratzer, Angelika. 1998. Scope or pseudoscope? Are there wide-scope indefinites? In Susan Rothstein (ed.), *Events and Grammar*, 163–196. Dordrecht: Springer.

Kurtzman, Howard S. & Maryellen C. MacDonald. 1993. Resolution of quantifier scope ambiguities. *Cognition* 48(3). 243–279.

Larson, Richard K. & Robert May. 1990. Antecedent containment or vacuous movement: Reply to Baltin. *Linguistic Inquiry* 21(1). 103–122.

Lee, So Young. 2019. A Minimalist parsing account of attachment ambiguity in English and Korean. *Journal of Cognitive Science* 3(19). 291–329.

Lee, Sunyoung. 2009. *Interpreting scope ambiguity in first and second language processing: Universal quantifiers and negation*. Manoa, HI: University of Hawai'i PhD dissertation.

Liu, Lei. 2018. Minimalist parsing of heavy NP shift. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, Hong Kong: Association for Computational Linguistics. `https://www.aclweb.org/anthology/Y18-1047`.

May, Robert. 1985. *Logical Form: Its Structure and Derivation*. Cambridge, MA: MIT Press.

Moulton, Keir. 2007. Scope relations and infinitival complements. University of Massachusetts Amherst, Ms.

Pasternak, Robert. 2019. Composing copies without trace conversion. Leibniz-Center for General Linguistics (ZAS), Ms.

Pereira, Fernando C.N. & David Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, 137–144. Cambridge, MA: MIT.

Rambow, Owen & Aravind K. Joshi. 1994. A processing model for free word order languages. In C. Clifton, L. Frazier & K. Rayner (eds.), *Perspectives on Sentence Processing*, 267–301. Mahwah, NJ: Lawrence Erlbaum Associates.

Reinhart, Tanya. 1997. Quantifier scope: How labor is divided between QR and choice functions. *Linguistics and Philosophy* 20(4). 335–397.

Rizzi, Luigi. 1978. A restructuring rule in Italian syntax. In Samuel Jay Keyser (ed.), *Recent transformational studies in European languages*, 113–158. Cambridge, MA: MIT Press.

Sag, Ivan. 1976. *Deletion and Logical Form*. Cambridge, MA: MIT PhD dissertation.

Stabler, Edward. 1997. Derivational minimalism. In Christian Retoré (ed.), *Logical aspects of computational linguistics* (vol. 1328 of *Lecture Notes in Computer Science*), 68–95. Berlin: Springer.

Stabler, Edward. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science* 5. 611–633.

Stabler, Edward & Edward Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science* 293(2). 345–363.

Sugawara, Ayaka, Hadas Kotek & Ken Wexler. 2013. Long vs. short QR: Evidence from the acquisition of ACD. In Sarah Baiz, Nora Goldman & Rachel Hawkes (eds.), *Proceedings of the 37th Boston University Conference on Language Development (BUCLD)*, 410–422. Somerville, MA: Cascadilla Press.

Syrett, Kristen. 2015a. Experimental support for inverse scope readings of finite-clause-embedded antecedent-contained-deletion sentences. *Linguistic Inquiry* 46(3). 579–592.

Syrett, Kristen. 2015b. QR out of a tensed clause: Evidence from antecedent-contained deletion. *Ratio* 28(4). 395–421.

Syrett, Kristen & Jeffrey Lidz. 2011. Competence, performance, and the locality of quantifier raising: Evidence from 4-year-old children. *Linguistic Inquiry* 42(2). 305–337.

Tanaka, Misako. 2015a. Asymmetries in long distance QR. In Anna E. Jurgensen, Hannah Sande, Spencer Lamoureux, Kenny Baclawski & Alison Zerbe (eds.), *Proceedings of the 41st Annual Meeting of the Berkeley Linguistic Society*, 493–501. Berkeley, CA: University of California, Berkeley Linguistic Society.

Tanaka, Misako. 2015b. *Scoping out of adjuncts: Evidence for the parallelism between QR and wh-movement*. London, UK: University College London PhD dissertation.

Tunstall, Susanne. 1998. *The interpretation of quantifiers: Semantics & processing*. Amherst, MA: University of Massachusetts Amherst PhD dissertation.

Wilder, Chris. 1997. Phrasal movement in LF: *de re* readings, VP-ellipsis and binding. In Kiyomi Kusumoto (ed.), *Proceedings of the North East Linguistic Society 27*, 425–439. Amherst, MA: Graduate Linguistic Student Association (GLSA), University of Massachusetts Amherst.

Wurmbrand, Susi. 2001. *Infinitives*. Berlin: Mouton de Gruyter.

Wurmbrand, Susi. 2014a. Restructuring across the world. In Ludmila Veselovská & Markéta Janebová (eds.), *Complex Visibles Out There. Proceedings of the Olomouc Linguistics Colloquium 2014: Language Use and Linguistic Structure*, 275–294. Olomouc: Palacký University.

Wurmbrand, Susi. 2014b. Tense and aspect in English infinitives. *Linguistic Inquiry* 45(3). 403–447.

Wurmbrand, Susi. 2015. Restructuring cross-linguistically. In Thuy Bui & Deniz Özyıldız (eds.), *Proceedings of the North East Linguistic Society 45*, 227–240. Amherst, MA: Graduate Linguistic Student Association (GLSA), University of Massachusetts Amherst.

Wurmbrand, Susi. 2018. The cost of raising quantifiers. *Glossa: a journal of general linguistics* 3(1): 19. 1–39.

Zhang, Chong. 2017. *Stacked relatives: their structure, processing and computation*. Stony Brook, NY: Stony Brook University PhD dissertation.