A computational implementation of a linear phase parser (2.x). Framework and technical documentation

Pauli Brattico

Research Center for Neurocognition, Epistemology and Theoretical Syntax (NETS)
School of Advanced Studies IUSS Pavia

Abstract

This document describes a computational implementation of a minimalist linear phase parser-grammar. The parser-grammar assumes that the core computational operations of narrow syntax (e.g., Merge/Move/Agree) are applied incrementally and on a phase-by-phase basis in language comprehension. Full formalization with a description of the Python implementation will be discussed, together with a few words towards empirical justification. The theory is assumed to be part of the human grammatical competence (UG).[1]

[1]Acknowledgements. This research has been conducted under the research project "ProGraM-PC: A Processing-friendly Grammatical Model for Parsing and Predicting Online Complexity" funded internally by IUSS (Pavia), PI Prof. Cristiano Chesi.

Table of Contents

1 Introduction

Language sciences distinguish linguistic competence from linguistic performance (Chomsky 1965). Competence refers to the knowledge of language possessed by native speakers. Any native speaker can answer questions such as “Is that sentence grammatical?” or “What does this sentence mean?” and, by doing so, provide a rich body of knowledge about his or her language. One might want to take a further step and examine also linguistic performance, namely how speakers and hearers put the knowledge in use in real-time communication. What kind of mistakes they make? How do they recover from errors? The latter inquiry presupposes the former: a theory of language use requires an explicit or implicit theory of the language whose use is being investigated. Competence, on the other hand, can be studied without taking performance into account.

It does not follow from this that linguistic performance must be irrelevant to the theory of competence. Whether performance should be considered when formulating a theory of competence can only be answered by an empirical inquiry. One strong (and hence interesting) form of a theory which assumes that performance should be taken into account when formulating a theory of competence is that of (Phillips 1996), who assumes that the theory of competence must incorporate a parser, a computational system that generates syntactic representations for the linguistic input available to the language hearer. Whether language comprehension/parsing is or is not part of the theory of linguistic competence, perhaps in the sense argued for by Phillips or in some other way, is an empirical question that cannot be settled by conceptual or methodological reasoning. I will argue, following Phillips, that certain aspects of performance do constitute an important ingredient of the theory of competence. The main hypothesis pursued in this study is that the “one-dimensional” properties of the PF-interface, when looked from the perspective of language comprehension, matter for the theory competence.

This document consists of two parts. Section 2 provides the empirical framework. These are assumptions that are in my view justified both by empirical and practical concerns; they constitute the hypothesis put forward in this document. Section 3 then goes into the details concerning full formalization and the computational implementation of that formalization. The claims made in these sections have a different status than those made in the first section. Many decisions that were made concerning full formalization could be done in several different ways, thus there is a certain degree of arbitrariness when it comes to many of the detailed assumptions concerning formalization. Furthermore, it is clear in retrospect that many assumptions concerning the full formalization can and will be further simplified in future work. The status of the claims presented in Section 3 and Section 3 therefore differ in terms of their epistemological relevance.

2 The framework

2.1 Language comprehension and the linguistic architecture

A native speaker can interpret sentences in his or her language by various means, for example, by reading sentences printed on a paper. Since it is possible accomplish this task in principle without any external information, it must be the case that all information required to interpret a sentence in one or several ways must be present in the sensory input. The operation that performs the required mapping from sensory objects into sets of possible meanings is called the *parser*, and the whole process that contains the parser is called *language comprehension*. Both are components of a still larger phenomenon, *language perception*, which included all linguistic sensory processes (e.g., listening to a foreign language, artificial voice generation, mechanical repetition). These are ultimately embodied as neuronal systems of the human brain that interact with each other and with other cognitive systems. When we judge sentences in our natural language to be grammatical or ungrammatical, we make use of the computational resources provided by these neuronal systems.

Some sensory inputs map into meanings that are hard or impossible to construct (e.g., *democracy sleeps furiously*), while others are judged by native speakers as outright ungrammatical. A realistic parser and a theory of language comprehension must appreciate these properties. The parser, when we abstract away from semantic interpretation, therefore defines a 'characteristic function' from sensory objects into a binary (in some cases graded) classification of the input in terms of its grammaticality or some other notion, such as semanticality or marginality. These categorizations are studied by eliciting responses from native speakers. Any parser that captures this mapping correctly will be said to be *observationally adequate*, to follow the terminology from (Chomsky 1965). Many practical parsers are not observationally adequate: they do not distinguish ungrammatical inputs from grammatical inputs. In fact, many have no reason to focus on ungrammatical sentences at all. A realistic, observationally adequate parser must nevertheless capture them correctly.

Some aspects of the parser are language-specific, others are universal and depend on the linguistic architecture and thus on the biological structure of the brain. A universal property can be elicited from the speakers of any language. For example, it is a universal property that an interrogative clause cannot be formed by moving an interrogative pronoun out of a relative clause (**who did John met the person that Mary admires_?*). On the other hand, it is a property of Finnish and not English that singular marked numerals other than 'one' assign the partitive case to the noun they select (*kolme sukkaa* 'three.sg.0 sock.sg.par'). The latter properties are acquired from the input in some way during language acquisition. The universal properties plus the storage systems constitute the fixed portion of the parser, whereas language-specific contents stored into the memory systems during language acquisition and other relevant experiences constitute the language-specific, variable part. A theory of parser that captures the fixed and variable components in a correct or at least realistic way without contradicting anything known about the process of language acquisition is said to reach *explanatory adequacy*.

The distinction between observationally adequate and explanatorily adequate parser can be appreciated in the following way. It is possible to design an observationally adequate parser for Finnish such that it replicates the responses elicited from native speakers, at least over a significant range of expressions, yet the same parser and its principles would not work in connection with a language such as English, not even when provided a fragment of English lexicon. We could design a different parser, using different principles and rules, for English. To the extent that the two language-specific parsers differ from each other in a way that is inconsistent with what is known independently about language acquisition, they would be observationally adequate but fall short of explanatory adequacy. An explanatory parser would be one that correctly captures the distinction between the fixed universal parts and the parts that can change through learning, given the evidence of such processes, hence such a parser would comprehend sentences in any language when supplied with the (1) fixed, universal components and (2) the information acquired from experience. We are interested in a theory of language comprehension that is explanatory.

Suppose we have constructed a theory of the parser that is arguably observationally adequate and explanatory, or at least tries to satisfy these requirements in a serious way. Then it is possible to ask a further question: does it agree with the data obtained from neuro- and psycholinguistic experimentation? Satisfying this requirement is not necessary, because realistic online parsing involves several features that an observationally adequate explanatory theory of the parser need not capture. One such property concerns automatization. Systematic use of language in everyday communication allows the brain to automatize the recognition and processing of linguistic stimuli in a way that an observationally adequate explanatory parser might or might not want to be concerned with. Furthermore, real language processing is sensitive to top-down and contextual effects that can be ignored when constructing a theory of the parser. That being said, it remains the case that the amount of computational resources consumed by the parser should be related in some meaningful proportion with what is observed in reality. If, for example, the parser engages in astronomical garden-path derivations when no native speaker exhibits such problems, then the parser can be said to be insufficient in its ability to mimic real language comprehension. Let us say that if the parser's computational efficiency matches with that of real speakers, the parser is also *psycholinguistically adequate*. I will adopt this criterion in this study as well.

Psycholinguistic adequacy is important also in the following sense. Suppose we build a generation theory which, instead of listening and parsing sentences, generates them. In fact, many theories currently in the literature are generative in that sense: they explain what word combinations are grammatical by showing what type of generations are possible. Suppose the theory is G. If G is formulated in a completely rigorous way, we can transform it trivially into a parser. The hypothetical parser G_P reads input sentences S and then check if S can be generated (from the same lexical resources) by G. The vice versa is also possible. Suppose we have a parser P, say the one delineated in this document. We can turn P into a generator P_G by letting it to generate word combinations freely while P checks if they are grammatical.[2] What this means is that if we ignore psycholinguistic adequacy, then parsing and generation are mathematically the same thing. But as soon as we take psycholinguistic facts into account, they diverge. For example, a theory in which the language faculty in the brain would check the grammaticality of any S by generating all sentences from the same lexical resources would be completely unrealistic. Here we are, on the other hand, guided by concerns of this type. On the other hand, nothing forces one to take any psycholinguistic facts into account; a pure parser-generator approach is as legitimate as one that incorporates additional constraints, especially if the parser-generator is also explanatory.

Recall that the parser, or a broader theory of language comprehension, must embody a theory of competence, but that it would be a mistake to infer from this truism that a theory of competence can ignore language comprehension without empirical justification. It is possible that the correct theory of the characteristic function must take language comprehension and parsing into account. This is trivially so if the two are based on the same system (or share a significant amount of computational and representational resources). It might even be that the language production side of the system has initially much more freedom in its operation and is literally constrained by the language comprehension system that imposes stricter conditions on what is acceptable and what is not. These questions cannot be settled a priori; they, too, are empirical.

But is it the case, then, that the same system performs language production and perception? Taken literally the claim is false. Language production utilizes motoric programs that allow the speaker to orchestrate a complex motoric sequence for the purposes of generating concrete speech or other forms of linguistic behavior; language comprehension involves perception and not necessarily concrete motoric sequencing. Although there is evidence that perception involves (or "activates") the motoric circuits and vice versa, overt repetition is (thankfully) not a requirement. A more interesting claim would be that the two systems share computational resources. This is the position taken in this study. Specifically, I will hypothesize, following (Phillips 1996, 2003), that many of the core computational operations involved in language production and/or in the determination of linguistic competence are also involved in language comprehension. The same could be true of lower-level language processing, so that the motoric generation of, say, phonemes is decoupled in the human brain with systems that are ultimately responsible for the perception of the same units. The idea is illustrated in Figure 10.

---

[2] These equivalences work if neither G nor P uses an infinite amount of computational resources, which is a reasonable assumption in this context.

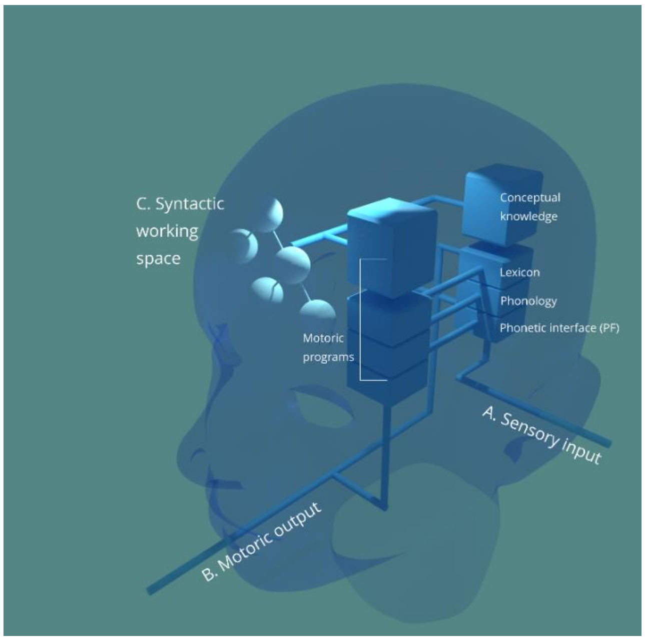

Figure 10. Language comprehension and production are based on the same system which, in turn, encapsulates the abstract knowledge of language (in essence, a function from sensory inputs into grammaticality judgment). The system is based on sensory mechanisms (posterior parts) and motoric mechanism (anterior parts) which work in tandem in both language production and comprehension, hence the connections between the systems. Both connect with a syntactic working space utilizing core computational information processing operations (Merge, Move, Agree). The syntactic working space seems to be realized by the neuronal circuits of the frontal cortex and its reciprocal connections to the subcortical regions. This architecture, although simplified, seems to me to be at least consistent with what we know today based on neurolinguistic research.

Language comprehension proceeds from concrete sensory properties into abstract meaning. This follows from the assumption that the process begins from sensory input and can operate successfully without additional input. Research shows that the process relies on increasingly more abstract properties of the input. Thus, once the sensory input is judged as linguistic (and not, say, music or ambient noise), it will be interpreted in terms of phonemes, syllables, surface morphemes and features, phonological words, lexical items, syntactic phrase fragments and whole expressions, and finally in terms of meaning and communicative intentions. More abstract interpretations and categories rely on those made at earlier stages. If this were not the case, then the system would assign a hallucinatory interpretation to the sensory input that has no basis in the input. This would correspond to an interpretation that is assigned either randomly, or no matter what to every input. Both are irrelevant, if they ever occur.

Certain properties of the language perception pipeline are necessary. The system must be able to recognize and decompose words. When a word is not recognized, it will be assigned, minimally, into a lexical category (noun, verb, and others). These words must then activate *lexical entries* that guide the parser in arranging them into appropriate syntactic and semantic representations. For example, the Finnish lexicon must determine that a finite verb agrees in number and person (and not in gender) with its subject, but a modal verb such as *täytyy* ‘must’ does not. Similarly, it must determine that the modal verb contains tense and takes a non-tensed A-infinitival verb as its complement, whereas a regular finite verb is complemented with a noun phrase or other object (depending on the verb). These lexical features allow the parser to arrange the words into reasonable or correct syntactic configuration. Yet, irrespective of whether we can determine or guess the meaning of the sentence without creating the corresponding syntactic configurations, a native speaker of Finnish will be able to say that a sentence in which these rules are violated is ungrammatical. For example, a native speaker of Finnish will be able to say that while (1)a is grammatical and an acceptable sentence, (b-d) are not.

(1)

a.	Sinun	täytyy	lähte-ä.	
	you.gen	must.T.3sg	leave-A/inf	
	’You must leave.’			
b.	*Sinun	täydy-t	lähte-ä.	(Agreement on the modal verb.)
	you.gen	must.T-2sg	leave-A/inf	
c.	*Sinun	täytyy	lähde-t.	(Wrong verb form.)
	you.gen	must.T.3sg	leave-2sg	
d.	*Sinun	täytyy	lähte-vän.	(Wrong verb form.)
	you.gen	must.T.3sg	leave-VA/inf	

Similarly, we known from neurolinguistic and psycholinguistic evidence that speakers react immediately to violations of this type. This requires that the speaker can create a syntactic representation or interpretation for the input sentence and, by examining the properties the resulting structure (or noticing its absence), is able to judge that the clause is ungrammatical and violates the syntactic rules of the language. Moreover, detecting of violations of this type is instant. Thus, the lexical entries activated by the incoming words are arranged by the human brain into a syntactic representation of some kind. To accomplish this, the brain must minimally use a computational operation, call it Merge (Chomsky 1992), that is able in principle to combine the incoming words (or lexical items) into larger units. Let us examine what kind of properties Merge must have in order to operate in this way.

2.2 Merge^{-1}

Linguistic input is received by the hearer in the form of sensory stimulus. That input can be thought as a one-dimensional string α, β, … of phonological words. We assume that in order to understand what the sentence means the human parser must create a set of abstract syntactic interpretations for the input string received through the sensory systems. These interpretations and the corresponding representations might be lexical, morphological, syntactic and semantic. One fundamental concern is to recover the hierarchical relations between words. To satisfy this condition, let us assume that while the words are consumed from the input, the core recursive operation in language, Merge (Chomsky 1995, 2005, 2008), arranges them into a hierarchical representation (Phillips 1996, 2003). For example, if the input consists of two words $\alpha + \beta$, Merge will yield $[\alpha, \beta]$(2).

(2) John + sleeps.

$$\begin{tikzcd} \text{John} \arrow[d] & \text{sleeps} \arrow[d] \\ {[\text{John},} & \text{sleeps]} \end{tikzcd}$$

The assumption that this process is incremental or "linear" means that each word consumed from the input will be merged to the phrase structure as it is being consumed. No word is "put aside for later processing." For example, if the next word is *furiously,* it will be merged with (2). There are three possible attachment sites, shown in (3), all which correspond to different hierarchical relations between the words.

(3) a. [[John *furiously*] sleeps] b. [[John, sleeps] *furiously*] c. [John [sleeps *furiously*]]

The operation illustrated in (3) differs in a number of respects from what constitutes currently the standard theory of Merge. When there is any danger of confusion, I will label the operation in (3) by Merge^{-1}, the symbol -1 referring to the fact that we are looking the operation from an "inverse perspective": instead of generating linear sequences of words by applying Merge, we are applying Merge^{-1} on the basis of a linear sequence of words and thus deriving the structure backwards.

Several factors regulate this process. One concern is that the operation creates a representation that is in principle ungrammatical and/or uninterpretable. Alternative (a) can be ruled out on such grounds: *John furiously* is not an interpretable fragment. Another problem of this alternative is that it is not clear how the adverbial, if it were originally merged inside subject, could have ended up as the last word in the linear input. The default linearization algorithm would produce **John furiously sleeps* from (3)a. Therefore, this alternative can be ruled by the fact that the result is ungrammatical and is not consistent with the word order discovered from the input. But what is consistent with the linear ordering?

If the default linearization algorithm proceeds recursively in a top-down left-right order, then each word must be merged to the "right edge" of the phrase structure, "right edge" referring to the top node and any of its right daughter node, granddaughter node, recursively. See Figure 2.

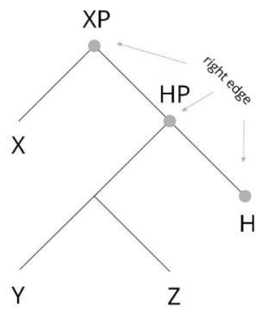

Figure 2. The right edge of a phrase structure. These three nodes are all possible attachment sites for an incoming word. X, Y and Z are not. This condition follows from the assumption that linearization in language production is always top-down/left-right.

This would mean either (b) or (c). (Phillips 1996) calls the operation "Merge Right". We are therefore left with the two options (b) and (c). The parser-grammar will select one of them. Which one? There are many situations in which the correct choice is not known or can be known but is unknown at the point when the word is consumed. In an incremental parsing process, decisions must be made concerning an incoming word without knowing what the remaining words are going to be. It must be possible to backtrack and re-evaluate a parsing decision made at an earlier stage. Let us assume that all legitimate merge sites are ordered and that these orderings generate a recursive search space that the parser-grammar use to backtrack. The situation after consuming the word *furiously* will thus look as follows, assuming an arbitrary ranking:

(4) *Ranking*

a. [[~~John *furiously*], sleeps]~~	(Eliminated)
b. 1. [[John, sleeps] *furiously*]	(Priority high)
c. 2. [John [sleeps *furiously*]]	(Priority low)

The parser-grammar will merge *furiously* into a right-adverbial position (b) and branch off recursively. If this solution will not produce a legitimate output, it will return to the same point and try solution (c). Every decision made during the parsing process is treated in the same way. We can depict this situation by considering that all of the possible solutions constitute 'potential phrase structures', which are ordered in terms of their ranking, while one of them is selected. This is illustrated in Figure 2.

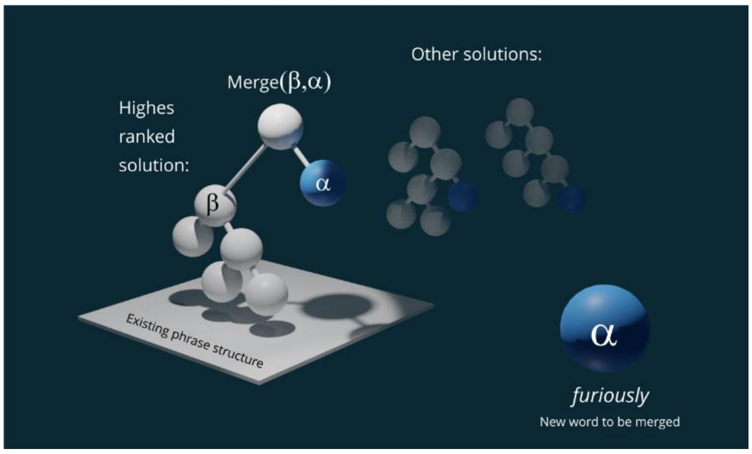

Figure 2. All possible merge sites of α on the right edge of the existing phrase structure β are ordered and then explored in that order. This operation takes place in the syntactic working space, see Figure 10.

Both solutions (a) and (c) are "countercyclic": they extent the phrase structure at its right edge, not at the highest node. A countercyclic Merge^{-1} is more complex than simple merge operation that combines to constituents: it must insert the constituent into a phrase structure and update the constituency relations accordingly (Section 3.4.1). This type of derivation could be called "top-down," because it seems to extend the phrase structure from top to bottom. The characterization is misleading: the phrase structure can be extended also in a bottom-up way, for example, by merging new items always to the highest node. It is more correct to say that merge is "to the right edge." In literal top-down grammars, such as that of (Chesi 2012), no bottom-up operation, to the right edge or to the left edge, is allowed.[3]

A final point that merits attention is the fact that merge right can break constituency relations established in an earlier stage. This can be seen by looking at representations (2) and (3)c, repeated here as (5).

```
(5)  John     sleeps                    furiously
       ↓        ↓                           ↓
     [John,   sleeps]   →   [John [sleeps   furiously]]
```

During the first stage, words *John* and *sleeps* are sisters. If the adverb is merged as the sister of *sleeps,* this is no longer true: *John* is now paired with a complex constituent [*sleeps furiously*]. If a new word is merged with [*sleeps furiously*], the structural relationship between *John* and this constituent is diluted further (6).

[3] Except for the root. The key empirical idea in restricted/literal top-down grammars is that every merge operation must be selected or expected "from above." That condition is too strong for parsing, which operates with a PF-object, but it might be still formulated as a condition for the LF-interface. The present model takes a weaker but also more general position, according to which top-down merge is possible but not the only option.

(6) [John [[sleeps furiously] γ]

This property of the Phillips architecture has several important consequences. One consequence is that upon merging two words as sisters, we cannot know if they will maintain a close structural relationship in the derivation's future. In (6), they don't: future merge operations broke up constituency relations established earlier and the two constituents were divorced. Consider the stage at which *John* is merged with the verb 'sleep' but with wrong form *sleep*. The result is a locally ungrammatical string **John sleep*. But because constituency relations can change in the derivation's future, we cannot rule out this step locally as ungrammatical. It is possible that the verb 'sleep' ends up in a structural position in which it bears no meaningful linguistic relation with *John*, let alone one which would require them to agree with each other. Only those configurations or phrase structure fragments can be checked for ungrammaticality that *cannot* be tampered in the derivation's future. Such fragments are called *phases* in the current linguistic theory (Chomsky 2000, 2001). I will return to this topic in Section 2.4. It is important to keep in mind that in the Phillips architecture constituency relations established at point t do not necessarily hold at a later point $t + n$.

There are at least two ways to think about what these changes mean to the overall linguistic architecture. The strong hypothesis, assumed by Phillips, is to say that parsing = grammar, hence that these are the true properties of Merge and nothing else is. We would then have to give up the standard properties of Merge that are postulated based on the bottom-up production theories (e.g., strict cyclicity). A weaker hypothesis is that the theory of Merge must be consistent with these properties. According to this alternative, Merge must be able to perform the computational operations described above (or something very similar), but we resist drawing conclusions concerning the production capacities that constitute the basis of standard theories of competence. The weaker hypothesis is less interesting than the strong one, and less parsimonious, but it makes it possible to pursue the performance perspective without rejecting linguistic explanations that have been crafted on the basis of the more standard production framework; instead, we try to see if the two perspectives can "converge" into some core set of assumptions or, at the very least, that they are not mutual contradictory. This is the position taken in this work. The operation of left-to-right Merge and its role in the whole language architecture is illustrated in Figure 11.

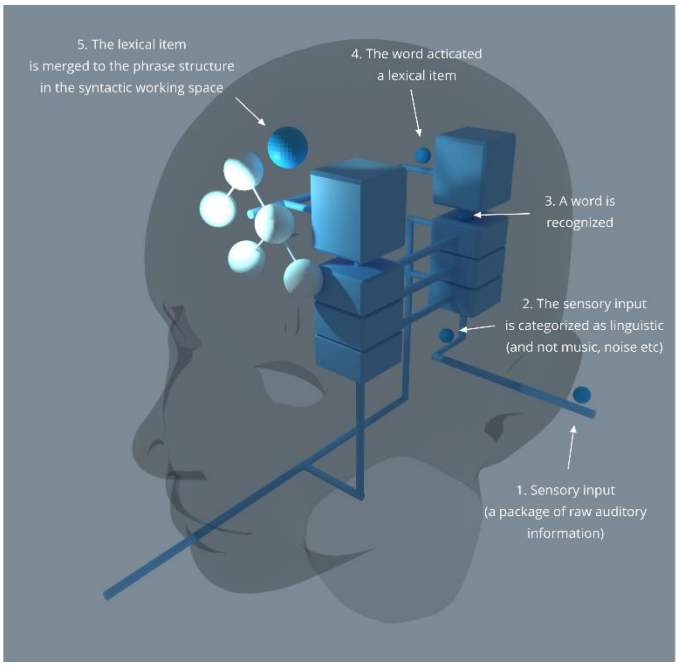
5. The lexical item is merged to the phrase structure in the syntactic working space
4. The word acticated a lexical item
3. A word is recognized
2. The sensory input is categorized as linguistic (and not music, noise etc)
1. Sensory input (a package of raw auditory information)

Figure 11. Merge and its role in language comprehension. Incoming sensory input is first analyzed through several subsystems until a phonological word is recognized. The phonological word is then decomposed into its primitive parts (if any), which are matched with lexical items in the lexicon. A lexical item (essentially a set of features) is then merged to the existing phrase structure in the syntactic working space.

It is not necessary to empower the parser with filtering and ranking. If they are not included, then the parser will explore all possible combinations of the input words. The result is a parser that can be observationally adequate and explanatory (to the extent that it will produce the correct classification and interpretation of the input sentences), but psycholinguistically vastly implausible. A parser that has no filtering or ranking function of any kind will typically use astronomical amount of computational resources when parsing a simple sentence. Therefore, filtering and ranking should be included if only for practical reasons. Once they have been included, it is then possible to test alternative ranking methods and match evaluate them against data from psycholinguistic experiments.

2.3 The lexicon and lexical features

Consider next a transitive clause such as *John admires Mary* and how it might be derived under the present framework (7).

```
(7)  John       admires  Mary
       ↓            ↓      ↓
     [John      [admires  Mary]]
```

There is much linguistic evidence that this derivation matches with the correct hierarchical relations between the three words (ignoring, of course, many details). The verb and the direct object form a constituent that is merged with the subject. We can imagine that this hierarchical configuration is interpretable at the LF-interface, with the usual thematic/event-based semantics: Mary will be the patient of the event of admiring, whereas John will be the agent. If we change the positions of the arguments, the interpretation changes. But the fact that the verb *admire*, unlike *sleep*, can take a DP argument as its complement must be determined somewhere. Let us assume that such facts are part of the lexical items: *admire*, but not *sleep*, has a lexical selection feature [!COMP:D] (or alternatively subcategorization feature) which says that it is compatible with, and in fact requires, a DP-complement. The idea has been explored previously by (Chesi 2004, 2012) within the framework of a top-down grammar, the key idea being that lexical features are ways to form and satisfy top-down "expectations." The fact that *admire* has the lexical feature [!COMP:D] can be used by Merge^{-1} to create a ranking based on an expectation: when *Mary* is consumed, the operation checks if any given merge site allows/expects the operation. In the example (8), the test is passed: the label of the selecting item matches with the label of the new arrival.

(8)	John	admires	Mary	
	↓	↓	↓	
	[John	[admires	Mary]]	
		[COMP:D]	D	*Mary* satisfies an expectation

Feature [COMP:L] means that the lexical item *licenses* a complement with label L, and [!COMP:L] says that it *requires* a complement of the type L. Correspondingly, [–COMP:L] says that the lexical item does *not* allow for a complement with label L. These features can be defined in a variety of ways.

There is a certain ambiguity in how these features are used. When the parser-grammar is trying to sort out an input, it will use lexical features to rank the solutions. These features cannot often be used to filter out possible solutions, because, as we just observed, the constituency relations can change in the derivation's future (Section 2.2). But when the phrase structure has been completed, and there is no longer any input to be consumed, the same features can be used for filtering purposes. This is because at this point, we know that no further rearrangements will take place. A functional head that requires a certain type of complement (say v-V) will crash the derivation if the required complement is missing. This procedure concerns features that are positive and mandatory (e.g. [!COMP:V]) or negative ([–COMP:V]). This filtering operation is performed at the LF-interface (discussed later) and will be later called an "LF-legibility test." Its function is to make sure that the phrase structure can be interpreted by the conceptual-intentional systems;; it is "perceived" as a meaningful linguistic unit. The crucial point is that no such filtering can be done locally while the first pass parse structure is being derived from the surface string.

Let us return to the example with *furiously*. What might be the lexical features that are associated with this item? The issue depends on the specific assumptions of the theory of competence, but let us assume something for the sake of the example. There are three options in (8): (i) complement of *Mary*, (ii) right constituent of *admires Mary*, and (iii) the right constituent of the whole clause. We can rule out the first option by assuming (again, for the sake of example) that a proper name cannot take an adverbial complement (i.e. *Mary* has a lexical selection feature [–COMP:ADV]). We are left with two options (9)a-b.

(9)
a. [[$_{S}$ John [admires Mary]] furiously]
b. [John [$_{VP}$ admires Mary] furiously]]

Independently of which one of these two solutions is the more plausible one (or if they both are equally plausible), we can guide Merge^{-1} by providing the adverbial with a lexical selection feature which determines what type of specifiers/sisters it is allowed or is required to have. I will call such features *specifier selection features*. A feature [SPEC:S] ("select the whole clause as a specifier/sister") favors solution (a), whereas [SPEC:V] favors solution (b). If the adverbial has both features, or has neither, then selection is arbitrary.

Notice that what constitutes a specifier selection feature will guide the selection of a possible left sister during parsing. Because constituency relations may change later, we do not know which elements will form specifier-head relations in the final parse. Therefore, we use specifier selection to guide the parser in selecting left sisters, and later use them to verify that the output contains proper specifier-head relations. To illustrate, consider the derivation in (10).

(10)
$$\begin{tikzcd}
\text{John's brother} \arrow[d] & \text{admires…} \arrow[d] \\
\text{[John's brother]} & \text{T(v, V)=finite tensed transitive verb}
\end{tikzcd}$$

The specifier selection feature of the finite transitive verb will instruct the parser to merge the finite transitive verb to the right of the subject *John's brother*. Notice that at the final output, after the processing of the direct object, the two are no longer sisters; instead, we will have the following configuration:

(11)
$$\begin{tikzcd}
\text{John's} & \text{brother} \arrow[d] & \text{admires} \arrow[d] & \text{Mary} \arrow[d] \\
\text{[[John's} & \text{brother]} & \text{[T(v,V)} & \text{DP]]}
\end{tikzcd}$$

The grammatical subject occurs in the canonical specifier position of T(v,V). It is important to keep in mind, once again, that in this framework most configurations established during first pass parsing are subject the change later in the parsing derivation.

2.4 Phases and left branches

Let us consider next the derivation of a slightly more complex clause (12).

(12) John's mother admires Mary.

↓ ↓ ↓ ↓

[$_{S}$[$_{DP}$ John's mother] [$_{VP}$ admires Mary]]

Notice that after the finite verb has been merged with the DP *John's mother*, no future operation can affect the internal structure of that DP. This follows from the assumption that merge is always to the right edge. All left branches become *phases* in the sense of (Chomsky 2000, 2001). This "left branch phase condition" was argued by (Brattico and Chesi 2019). We can formulate the condition tentatively as (13), but a more rigorous formulation will be given as we proceed.

(13) *Left Branch Phase Condition (LBPC)*
Derive each left branch independently.

The intuitive content of this principle is that we "throw all left branches away" from the active working space once have been assembled and fully processed (14).

(14) John's + mother + admires + Mary
[John's mother]
xxxxxxxxxxxxxxxxxx + admires + Mary

If no future operation is able to affect a left branch, as assumed in (13), then all grammatical operations (e.g. movement reconstruction) that must to be done in order to derive a complete phrase structure must be done to each left branch before they are sealed off. Furthermore, if after all operations have been done the left branch fragment remains ungrammatical or uninterpretable, then the original merge operation that created the left branch phase must be cancelled. This limits the set of possible merge sites. Any merge site that leads into an ungrammatical or uninterpretable left branch can be either filtered out as either unusable or be ranked lower.

The notion that each left branch phase is processed independently, and that all grammatical operations are applied to each left branch phase, requires a comment. A parser implements a function from PF-objects into sets of LF-objects, the latter which contains semantically interpretable phrase structures. Therefore, each left branch must be transferred from the active syntactic working space to the LF-interface, and from the LF-interface, if the representation is well-formed, into the conceptual-intentional system in which it is provided a semantic interpretation. The process that maps a candidate left branch phase into the LF-interface is called *transfer*. It involves several mechanical, reflex-like operations that normalize the phrase structure for semantic interpretation, for example, by reconstructing operator movement, so that each argument may receive a thematic role in addition to marking the scope an operator. These processes are discussed later, but the general architecture is illustrated in Figure 5.

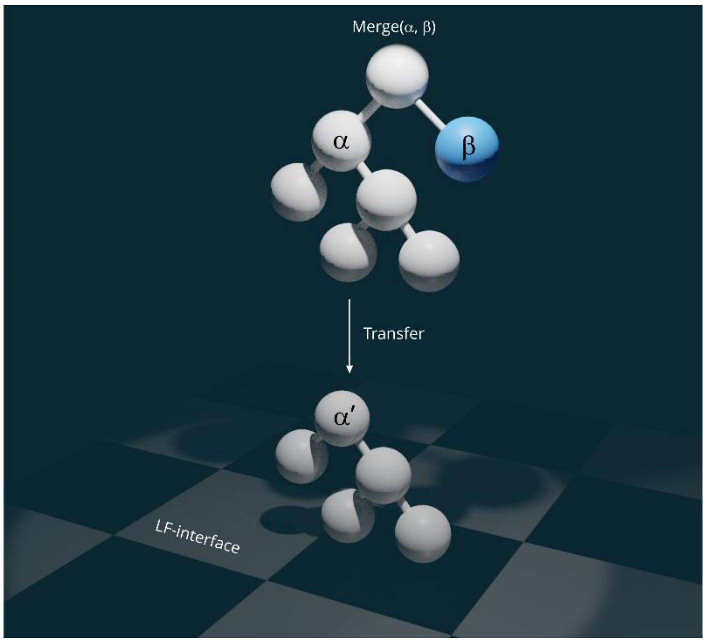

Figure 5. Transfer transforms a candidate left branch phrase α to the LF-interface. The transfer operation implements several computational operations, and the resulting phrase structure α´ is then evaluated at the LF-interface for semantic legibility. If the phrase structure is legit, it will be passed on to the conceptual-intentional systems for semantic interpretation. Transfer will be modeled here as a normalization or an error correction/tolerance algorithm,

The nature of the LF-interface is not trivial. It is typically conceptualized as the "interface" between syntax and semantics. One condition is that the syntactic representation occurring at the LF-interface must be such that it can be interpreted by the conceptual-intentional (semantic systems) in the way a native speaker interprets the sentence. Something is wrong if the object arriving at the LF is assigned an interpretation that does not match with the interpretation provided by the native speaker. Once the representation passes at LF, it will be handed over to extrasyntactic systems and it disappears from syntax. I do not view LF objects as a "language thought" that must be utilized in all acts of conceptual thinking, or as a universal conceptual starting point for the generation of linguistic expressions. There are several reasons for this. One is that many aspects of any given expression's meaning, such as the concrete references assigned to many pronouns, are determined by factors that are external to the LF-interface object. The second and more interesting reason is that there are aspects of semantic interpretation, "surface interpretations," that are determined by configurations that are generated before LF-interface. The LF-interface is more like an internal perceptual interface, in that it evaluates in a reflex-like manner whether the resulting linguistic object make sense semantically.

2.5 Labeling

Suppose we reverse the arguments (12) and derive (15).

(15)	Mary	admires	John’s	mother
	↓	↓	↓	↓
	Mary	[admires	[John’s	mother]]
		[COMP:D]		

The verb *admires* selects for a DP-argument, but the complement selection feature refers to the label of the complement phrase. The system must figure out the label of every complex constituent, such as *John’s mother*. A recursive labeling algorithm (16) is postulated for this purpose. The intuitive motivation of the algorithm is to search for the closest possible primitive head from the phrase, which will then constitute the label. Here “closest” means closest form the point of view of the selector; if we look at the situation from the point of view of the labeled phrase itself, then closest is the “highest” or “most dominant” head.

(16) Labeling

Suppose α is a complex phrase. Then

a. if the left constituent of α is primitive, it will be the label; otherwise,

b. if the right constituent of α is primitive, it will be the label; otherwise,

c. if the right constituent is not an adjunct, apply (16) recursively to it; otherwise,

d. apply (16) to the left constituent (whether adjunct or not).

The term *complex constituent* is defined as a constituent that has both the left and right constituent; if a constituent is not complex, it will be called *primitive constituent.* A constituent that has only the left or right constituent, but not both, will be primitive, according to these definitions. Conditions (16)c-d mean that labeling – and hence selection – ignores right adjuncts. This is used in this study as a defining feature of an “adjunct.”[4] Consider again the derivation of (2), repeated here as (17).

(17) John + sleeps.

↓ ↓

[John, sleeps]

If *John* is a primitive constituent having no left or right daughters, as seems to be the case, the labeling algorithm will label [*John sleeps*] as a DP. The primitive left constituent D will be its label, which is wrong. One solution is to reconsider the labeling algorithm. That seems implausible: (16) captures what looks to be a general property of language, thus this alternative would require us to treat (17) and other similar examples as exceptions. They are not exceptions, however: there is nothing anomalous about (17). The representation

[4] If both the left and right constituents are adjuncts, the labeling algorithm will search the label from the left. This is a slightly anomalous situation, which we might consider ruling out completely. The idea was tested by concrete simulation but was rejected. It leads into empirically correct results in some cases and is therefore preserved.

should come out as a VP, with the proper name constituting an argument of the VP of its head. Thus, the proper name should not constitute the (primitive) head of the phrase. There are two way to accomplish such outcome. One possibility is to redefine the notions of “primitive” and “complex” constituents in a way that would make proper names complex constituents despite having no constituents. If *John* were a complex constituent, then the labeling algorithm would take the (primitive) verb as the label. But the idea that *John* is a “complex constituent” despite having no constituents, left or right, suggest that we are confusing something. It is after all the case that *John* does not have any constituents. A more plausible alternative would be to define the labeling algorithm in relation to another property that correlates with the distinction between primitive and complex constituents. We could say that *John* is a primitive constituent that has a special property of, say, “MAX” that makes is unable to project further; labeling would then respond to this property instead of phrase structural complexity. This is a theoretical possibility, but unilluminating. It hides the real reason why *John* does not project while *sleep* does. In addition, the assumption is not innocent in the context of parsing, because it requires that the parser-grammar knows, of each element it processes, whether it is MAX or not MAX. If the property is mysterious, we must stipulate it for every constituent; or even worse, it could make every item ambiguous for the parser.

For these reasons, I suggest we keep the definition of labeling as provided and assumed that *John* is a complex constituent despite of appearing in the surface string as if it were not. I assume that its structure is [D N], with the N raising to D to constitute one phonological word. This information can only come from the lexicon/morphological parser, in which proper names are decomposed into D + N structure. The structure of (17) is therefore (18), with “D” and “N” coming from the lexicon/morphology.

(18)
$$\begin{tikzcd}
\text{John +} \arrow[d] & & \text{sleeps.} \arrow[d] \\
\text{D} \arrow[d] & \text{N} \arrow[d] & \text{sleeps} \arrow[d] \\
{[_{VP}[_{DP}\text{D}} & {\text{N}]} & {\text{sleeps}]}
\end{tikzcd}$$

It is important to note that the labeling algorithm (16) presupposes that primitive elements, when they occur in prioritized (i.e., left) positions, always constitute heads. A head at the right constitutes a head if there is a phrase at left. This happens in (18), which means that the whole phrase will come out as a verb phrase. The outcome will be the same if the verb is transitive. The structure and label are provided in (19).

(19) $[_{VP}[_{DP}\text{ D N}]\ [\ V^0\ [_{DP}\text{ D N}]]$

This analysis presupposes that there is some way to unpack *John* into the complex constituent [D N]. That operation is part of transfer (Section 2.8.3).

2.6 Adjunct attachment

Adjuncts, such as adverbials and adjunct PPs, present a separate problem. Consider (20).

(20)

a.	*Ilmeisesti*	Pekka	ihailee	Merjaa.
	apparently	Pekka	admires	Merja
b.	Pekka	*ilmeisesti*	ihailee	Merjaa.
	Pekka	apparently	admires	Merja
c.	Pekka	ihailee	*ilmeisesti*	Merjaa.
	Pekka	admires	apparently	Merja
d.	??Pekka	ihailee	Mejaa	*ilmeisesti*
	Pekka	admires	Merja	apparently

The adverbial *ilmeisesti* 'apparently' can occur almost in any position in the clause. Consequently, it will be merged into different positions in each sentence. This will confuse labeling and structure-building, creating vastly different structural interpretations for each sentence, whereas in truth it is clear that they all share the same basic SVO structure while the adverbial is attached 'freely' into any position in such a way that it does not interfere with labeling. The problem is to define what this type of 'free attachment' means.

It is assumed here that adjuncts are geometrical constituents of the phrase structure, but they are stored in a secondary syntactic working memory and are invisible for sisterhood, labeling and selection in the main working memory. Thus, the labeling algorithm specified in Section 2.5 ignores adjuncts. The result is that the label of (21) is V: while analyzing the higher VP shell, the search algorithm does not enter inside the adverb phrase; the lower VP is penetrated instead (16)c-d.

(21) H^0 [$_{\text{VP}}$ John [$_{\text{VP}}$[$_{\text{VP}}$ admires Mary] ⟨$_{\text{AdvP}}$ furiously⟩]]

The reasoning applies automatically to selection by the higher head H: if H has a complement selection feature [COMP:V], it will be satisfied by (21). Consider (22).

(22) John [sleeps ⟨$_{\text{AdvP}}$ furiously⟩]

The adverb constitutes the sister of the verbal head V^0 and is potentially selected by it. This would often give wrong results. This unwanted outcome is prevented by defining the notion of sisterhood so that it ignores right adjuncts. The adverb *furiously* resides in a secondary syntactic working memory and is only loosely (geometrically) attached to the main structure; the main verb does not see it in its complement position at all. From the point of view of labeling, selection and sisterhood, the structure of (22) is [$_{\text{VP}}$[*John*] *sleeps*]. The reason adjuncts must constitute geometrical parts of the phrase structure is because they can be still targeted by several syntactic operations, such as movement and case assignment (Agree). Adjunct attachment is illustrated by Figure 3.

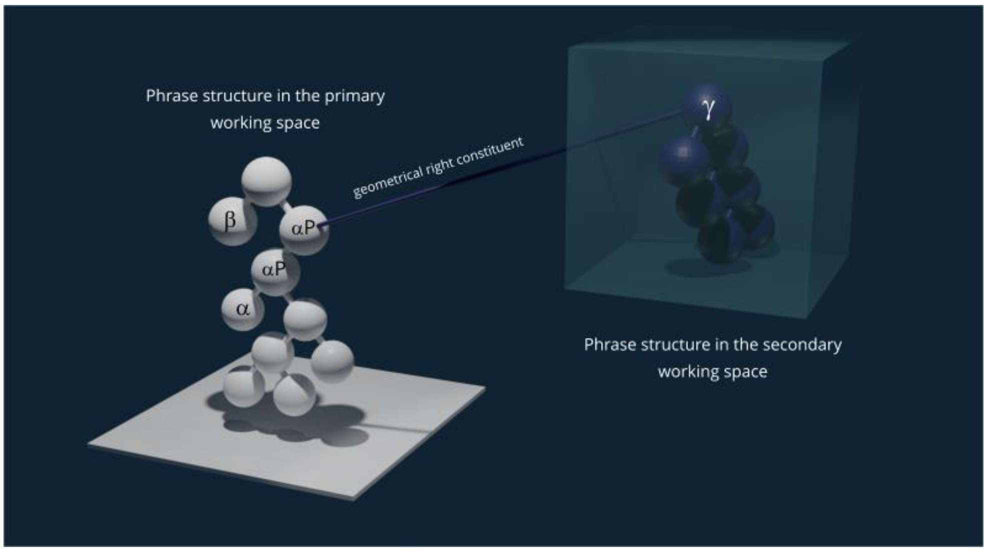
Phrase structure in the primary working space
geometrical right constituent
γ
β
αP
αP
α
Phrase structure in the secondary working space

Figure 3. Adjuncts γ are attached to the phrase structure as geometrical constituents, but they are "pulled" into a secondary syntactic working space. Being in the second working space, they are invisible for the primary computational operations (sisterhood, selection, labeling) inside the primary working space. Element β therefore sees αP as its sister. However, they are still part of the phrase structure and can be targeted for case assignment and movement.

The fact that adjuncts are optional is explained by the fact that they are automatically excluded from selection and labelling: whether they are present or absent has no consequences for either of these dependencies. This explains also the fact that their number is not limited.

These assumptions entail that adjuncts can be merged anywhere, which is not correct. I assume that each adverbial (head) is associated with a feature *linking* or *associating* it with another label or feature. The linking relation is established by means of an 'inverse probe-goal relation' that I call *tail-head relation*. For example, a VP-adverbial is linked with V, a TP-adverbial is linked with T, a CP-adverbial is linked with C. Linking is established by checking that the adverbial (i) occurs inside the corresponding projection (e.g., VP, TP, CP) or is (ii) c-commanded by a corresponding head (23). This theory owes much to (Ernst 2001), who likewise assumes that adjuncts do not have fixed canonical positions.

(23) Condition on tail-head dependencies

A tail feature F of head H [TAIL:F] can be checked if either (a) or (b) holds:

a. H occurs inside a projection whose head has F;

b. H is c-commanded by a head whose head has F.

Condition (i) is uncontroversial. Condition (ii) allows adverbials to remain in a low right-adjoined or extraposed positions in a canonical structure. If condition (ii) is removed from the model, then all adverbials will be reconstructed into positions in which they are inside the corresponding projections (reconstruction will be discussed later). A VP-adverbial will be reconstructed inside a VP, and so on. Some versions of the theory keep (ii), others deny it; the matter is controversial. The tail-head dependency is illustrated in Figure 4.

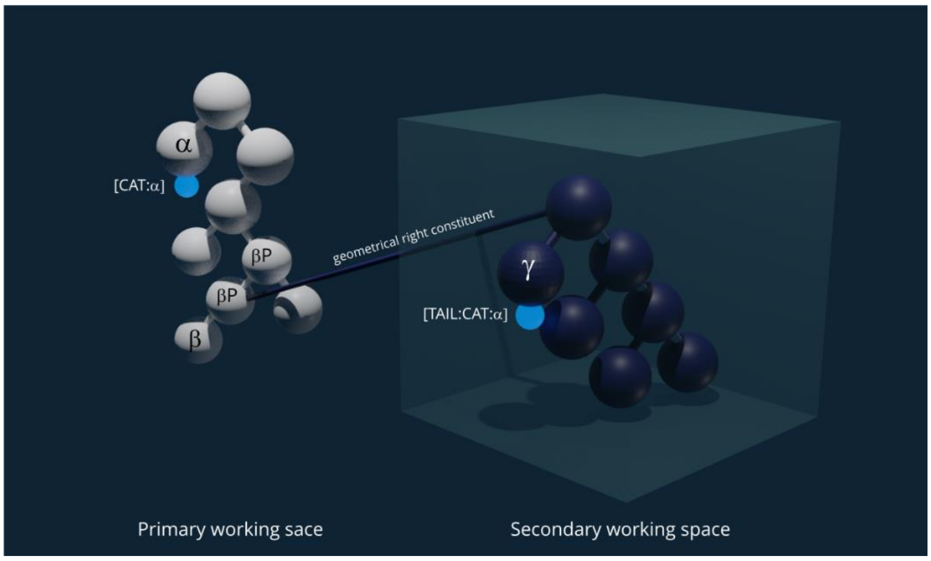

Figure 4. Adjuncts must satisfy a tail-head relation. The head γ of the adjunct with the tail-head feature [TAIL:CAT:α] must be c-commanded by a head that has feature [CAT:α]. In the more traditional bottom up theories, [TAIL:CAT:α] is the goal feature requiring checking, whereas [CAT:α] is the probe feature. Another legitimate condition is if the probe feature were at β, because the adjunct would then be inside βP.

If an adverbial/head does not satisfy a tail feature, it will be reconstructed into a position in which it does during Transfer. This operation will be discussed in Section 2.8.4.

2.7 EPP and “unselective” selection

Some languages, such as Finnish and Icelandic, require that the specifier position of the finite tense is filled in by some phrase, but it does not matter what the label of that phrase is. This is captured by unselective specifier features [SPEC:*] and [!SPEC:*]. These features check that a phrase of any label (hence *) fills in the specifier position of the lexical element [SPEC:*], or requires it [!SPEC:*]. Because the feature is unselective, it is not interpreted thematically, it cannot designate a canonical position, and hence the existence of this feature on a head triggers A´/A movement reconstruction (Section 2.8.1). This constitutes a sufficient (but not necessary) feature for reconstruction; see 2.8.1. Language uses phrasal movement, and hence an unselective specifier feature, to represent a null head (such as C) at the PF-interface. Corresponding to [SPEC:*] we also have [!COMP:*], which is the property all functional heads have by definition.

2.8 Move^{-1}

2.8.1 A-bar reconstruction

A phrase or word can occur in a canonical or noncanonical position. These notions get a slightly different interpretation within the comprehension framework. A canonical position *in the input* could be defined as one that leads the parser-grammar to merge the constituent directly into a position at which it must occur at the LF-interface. For example, regular referential or quantificational arguments must occur inside the verb phrase at the LF-interface in order to receive thematic roles and satisfy selection. Example (24) illustrates a sentence in which all elements occur in their canonical positions in the input: the parser reaches a plausible output by merging the elements into the right edge as they are being consumed.

(24)	John	admires	Mary
	↓	↓	↓
	[John	[admires	Mary]]

Example (25) shows a variation in which this is no longer the case.

(25)	Ketä	Pekka	ihaile-e	__?	(Finnish)
	who.par	Pekka.nom	admire-3sg		
	‘Who does Pekka admire?’				

The parser will generate the following first pass parse for this sentence (in pseudo-English for simplicity):

(26) [who [Pekka admires]]

The clause violates selection at least twice: first due to the occurrence of the two specifiers DP_1 and DP_2 at the left edge and then due to the missing object of *admire*. The two violations are related: the element that triggers the double specifier violation at the left edge is the same element that is missing from the complement position. Therefore, it is obvious that the interrogative pronoun causes the double specifier error because it has been “dislocated” to the wrong position from its canonical complement position, and this is the reason it also triggers the complement selection failure. The parser must reverse-engineer dislocation in order to create a representation that can be interpreted at the LF-interface. These operations take place during transfer. The double specifier problem is first fixed by generating a head between them (27).

(27)	[Ketä	[C(wh)	[Pekka	ihailee	__]]]?
	wh	C(wh)			
	who.par		Pekka	admires	
	‘Who does Pekka admire?’				

The mechanism has two computational steps. The first is the recognition that a head is missing. That is inferred from the existence of the two specifiers of the T/fin head (two specifiers are not allowed unless the head as a specified feature licensing them). The next step is to generate the label for the head. This information is obtained from the criterial feature. This allows the parser-grammar to infer both the existence and the nature of the phonologically null head. The interrogative phrase must be reconstructed back to its canonical LF-position to satisfy the complement selection for the verb *admire*.

(28) Who does John admire __?
$\xrightarrow{\text{Reconstruction}}$

Reconstruction (also called Move^{-1} in this paper) works by copying the element in the wrong position and by *dropping* it into the same structure, in this case dropping begins from the sister of the targeted element and proceeds downward while ignoring left branches (phases) and right adjuncts. The element is copied to the first position in which (1) it can be selected, (2) is not occupied by another legit element, and in which (3) it does not violate any other condition for LF-objects. If no such position is found, the element remains in the original position and may be targeted by another operation later during transfer. If the position is found, it will be copied there; the original element will be tagged so that it will not be targeted for the second time.

Move^{-1} takes place during transfer. Transfer itself is triggered under two conditions. One condition is that all words have been consumed, in which case the final product will be transferred to LF. The second condition occurs upon Merge^{-1}(α, β) and leads to the transfer of α. These two conditions are expressed by (29)(Brattico and Chesi 2019).

(29) Merge(α, β) or α is completed $\leftrightarrow$ Transfer(α)

The reason α is and must be transferred is that we want to check if it constitutes a legitimate LF-object. If not, the fragment/sentence can be rejected. Thus, if we are considering Merge^{-1}(α, β) as a possible merge site for β, failed transfer would signal for the parser that the solution should be filtered out or ranked lower. If α is complete and transfer fails, then the parser must backtrack. It remains to argue that there is no need to perform additional transfer operations. The problem with such a strategy is that any other fragment can still be repaired by consuming more words from the input and merging them to the structure, hence transfer would be useless.

One question that remains without explicit answer is the explicit trigger for the operation that reconstructs the interrogative pronoun to the complement position. Notice that after C(wh) has been generated to the structure, all selection features are potentially checked. One possibility is that the operation is triggered by the missing complement. That is, the system will recognize that the complement is missing, and then searches for a suitable filler. In the present implementation, it is assumed that error recovery is triggered at the site of the element that needs reconstruction. In this case, it is assumed that C(*wh*) has an unselective specifier selection feature [SPEC:*] (=generalized EPP feature, or second edge feature in the sense of (Chomsky 2008)) which says that the element may have a specifier with any label that is not selected (in the present formalization, labels select and are selected). Reconstruction is triggered by the presence of this feature (Brattico and Chesi 2019). The reconstructed canonical position will be found during dropping.

It might be assumed that transfer represents the inverse of production/generation procedure. I do not think that it is used in this way. A more likely hypothesis, elucidated later, is that transfer is part of a broader error recovery mechanism. The production side might be relatively or completely free and only constrained by the input-output relations established by the comprehension algorithm, including transfer.

2.8.2 A-reconstruction and EPP

As pointed out above, one sufficient condition for phrasal reconstruction is the occurrence of a phrase at the specifier position of a head that has the [SPEC:*] feature (=EPP in the standard theory). This alone will trigger reconstruction, with or without criterial features. This creates A-movement reconstruction under the present system.

2.8.3 Head reconstruction

Many heads occur in noncanonical positions in the input string. Consider (30).

(30)	Nukku-a-ko	Pekka	ajatteli	että	hänen	pitää	_?
	sleep-T/inf-Q	Pekka	thought	that	he	must	

‘Was it sleeping that Pekka thought that he must do?’

The complex word *nukkua-ko* ‘sleep-T/inf-Q’ contains elements that are in the wrong place. The infinitival verb form (T/inf, V) cannot occur at the beginning of a finite clause, and there is again an empty position at the end of the clause in which a similar element is missing.

Lexical and morphological parser provides the parser-grammar with the information that the -kO particle in the first word encodes the C-morpheme itself (C(-kO), which is then fed into the parser-grammar together with the rest of the morphological decomposition of the head. In this case, the verb *nukkua-ko* is composed out of C(-*kO*), infinitival T_{in} (-*a*-) and V (*nukku*-). Morphology extracts this information from the phonological word and feeds it to syntax in the order illustrated by (31). Symbol “#” indicates that there is no word boundary between the morphemes/features.

(31)	C(-kO) +	#T_{inf} +	#V +	Pekka +	ajatteli +	että +	hänen +	pitää
	C	T_{inf}	V	Pekka	thought	that	he	must

The individual heads are then collected into one complex head by the syntactic parser. The incoming morphemes are “stored” into the right constituent of the preceding morpheme, a process that resembles cliticalization.[5] Thus, if syntax receives a word-internal morpheme β, it will be merged to the right edge of the previous morpheme α: [$_{\alpha}$ ∅ β]. Notice that by defining “complex constituent” as one that has both the left and right constituent, [$_{\alpha}$ ∅ β] comes out as primitive constituent and can therefore constituent the head of a projection. The linear sequence C-T/inf-V becomes [$_{\mathrm{C}}$ ∅ [$_{\mathrm{T/inf}}$ ∅ V]], and this is what gets originally merged (32).

(32) [$_{\mathrm{C}}$ ∅ [$_{\mathrm{T/inf}}$ ∅ V]]	Pekka	ajatteli	että	hänen	pitää	__.
	Pekka	thought	that	he	must	

Representation (32) is not a legit LF-object. The first constituent [$_{\mathrm{C}}$ ∅ [$_{\mathrm{T/fin}}$ ∅ V]] cannot be interpreted, and the embedded modal verb *pitää* ‘must’ lacks a complement. Both problems are solved by head reconstruction which drops [$_{\mathrm{T/inf}}$ ∅ V] from within C into the structure. This is done by finding the closest position in which T/inf can be selected and in which it does not violate local selection rules. The closest possible position for T/inf is the empty position inside the embedded clause. V will then be extracted in the same way and reconstructed into the complement position of T$_{\mathrm{inf}}$.

(33) [$_{\mathrm{C}}$ ∅ [$_{\mathrm{T/inf}}$ ∅ V]]	Pekka	ajatteli	että	hänen	[pitää	[[$_{\mathrm{T/inf}}$ ∅ V]	V]].
	Pekka	thoughr	that	he.gen	must		

$\vdash\!\!\xrightarrow{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \xrightarrow{\qquad\quad}$

The operation fixes problems with cluttered heads in the input. It resembles phrasal movement reconstruction: it considers the cluttered components of the complex head to be in a wrong position and attempts to drop them into first positions available in which they could create a legitimate LF-object.

2.8.4 Adjunct reconstruction

Consider the pair of expressions in (34) and their canonical derivations.

(34)

a.	Pekka		käski	meidän	ihailla	Merjaa.
	Pekka.nom		asked	we.gen	to.admire	Merja.par
	↓		↓	↓	↓	↓
	[Pekka	[	asked	[we	[to.admire	Merja]]]]
	’Pekka asked us to admire Merja.’					
b.	Merjaa		käski	meidän	ihailla	Pekka.
	Merja.par		asked	we.gen	to.admire	Pekka.nom

[5] Pronominal clitics are right/left constituents that are complex but occur without a sister.

↓ ↓ ↓ ↓ ↓

[Merja [asked [we [to.admire Pekka]]]

'Pekka asked us to admire Merja.'

Derivation (a) is correct, (b) is incorrect. This is because native speakers interpreted the thematic roles identically in both examples. The subject and object are again in wrong positions, and the sentence must be fixed. Yet, neither A´- nor A-reconstruction can handle these cases. The problem is created by the grammatical subject *Pekka* 'Pekka.nom', which has to move 'upwards' in order to reach the canonical LF-position Spec,VP. There is no operation that achieves this.

Because the distribution of thematic arguments in Finnish is very similar to the distribution of adverbials, I have argued that richly case marked thematic arguments can be promoted into adjuncts (Brattico 2016, 2018, 2019). See (Baker 1996; Chomsky 1995: 4.7.3; Jelinek 1984) for similar hypothesis. This can be captured in the following way (from (Brattico 2019)). Suppose that case features must establish local tail-head (inverse probe-goal) relations with functional heads as provided, tentatively and for illustrative purposes only, in (35).

(35) Case features must establish local tail-head relations such that

a. [NOM] is checked by +FIN,

b. [ACC] is checked by v (ASP),

c. [GEN] is checked by –FIN

d. [PAR] is checked by –VAL.

This means simply that case forms are morphological reflexes of tail-head features. The symbol "–VAL" refers to a head that never exhibits φ-agreement, but what the features are is not crucial here; what matters is that case features are associated with c-commanding functional heads and features therein. If the condition is not checked by the position of an argument in the input, then the argument is treated as a an adjunct and reconstruction (dropped) into a position in which (35) is satisfied. In this way, the inversed subject and object can find their ways to the canonical LF-positions (36). Notice that because the grammatical subject *Pekka* is promoted into adjunct, it no longer constitutes the complement of the verb; the partitive-marked direct object does.

(36) [⟨Merjaa⟩$_2$ T/fin [__$_1$ [käski [meidän [ihailla [__$_2$ ⟨Pekka⟩$_1$]]]]

+FIN ⟵[nom] –VAL ⟵[par]

Merja.par asked we.gen to.admire Pekka

'Pekka asked us to admire Merja.'

The operation is licensed by rich case morphosyntax, which allows transfer to dislocate the orphan arguments to their correct canonical positions. In a language with richer morphosyntax, many more errors in word order are possible than in a language in which morphosyntax is more meager. Case features are an extension of the tail-head dependency feature introduced earlier in Section 2.6.

2.8.5 Ordering of operations

All movement reconstruction is done during Transfer. The operations must be ordered. Both A´/A-reconstruction and adjunct reconstruction presuppose head reconstruction, because it is only the presence of heads and their lexical features that can guide A´/A- and adjunct reconstruction. The former relies on EPP features and empty positions, whereas the latter relies on the presence of functional heads. Furthermore, A´/A-reconstruction relies on adjunct reconstruction: empty positions cannot be recognized as such unless orphan constituents that might be hiding somewhere are first returned to their canonical positions. The whole sequence is (37).

(37) Merge^{-1}(α, β) $\rightarrow$ Transfer α (reconstruct heads $\rightarrow$ reconstruct adjuncts $\rightarrow$ reconstruct A/A'-movement)

The whole sequence is performed in a one fell sweep, in a reflex-like manner; it is not possible to evaluate the operation only partially or backtrack some moves while executing others.

2.8.6 Movement as error tolerance

The following input condition turns out to be virtually universal when movement reconstruction/transfer is examined from the point of view of language comprehension:

(38) Input condition (nontechnical definition)

Trigger an operation during transfer if and only if the first pass parse has an element α in a position β that will not satisfy LF-legibility.

The operation has the general character of an 'error tolerance mechanism' that attempts to fix a problem in the first pass parse. The output condition of transfer will almost always satisfy (39):

(39) *Output condition (nontechnical definition)*

Transfer will attempt to generate a legitimate LF-object.

The hypothesis that transfer performs error correction, and that speaker use it intentionally, might lead one to assume that the non-normalized surface structure is irrelevant to semantic interpretation and constitutes mere "noise." That this is not true is well-known: there are several surface interpretation effects. In Finnish and other languages in which word order is relatively free, the noncanonical positions are often associated with information structure interpretation (e.g., topic and focus interpretations). Furthermore, there are operations that are sensitive to surface scope. One possibility is to associate the various derivational stages occurring during transfer with independent semantic interpretation mechanisms. The second alternative is to made LF-interpretation sensitive to different occurrences (copies) of the elements. Of these two alternatives, the first is more principled in the sense that it requires very few assumptions that are completely artificial, and it explains why speakers use the error tolerance device intentionally, namely, in order to generate the secondary "surface" interpretations.

All or most operations performed during transfer discussed in the preceding sections follow the same general logic. An error is recognized by detecting a selection violation and/or by noticing that the element is c-commanded by an illegitimate head (probe-goal violation). These properties are detected from the first pass parse structure (s-structure). Error detection is followed by 'dropping', in which the illegitimate object α is dropped into the phrase structure by fitting it to any position in the structure, starting from the top, and in which the first legitimate position is selected. The operation tries to find a proper position for the orphan element and adopts the first such position. Dropping cannot visit left branches, which are phases (and hence already out of the system), and it cannot visit right adjuncts, which are invisible. A constituent may also be turned into adjunct, however, which results in "extraposition." This is often used as a last resort option.

Criterial features are used to generate phonologically null heads and/or to generate their features. Here error recovery is trying to generate an element that is missing from the PF-interface object by using features form another local element. The operation is triggered by the presence of criterial feature at a phrase; no selection or probe-goal violations need to be present. For example, if the grammatical subject contains a *wh*-feature, that feature will be automatically copied to the local T/fin head to result in an interrogative finite tense head. The system is viewing the criterial feature as a 'virus feature' whose presence is not motivated; if so, then also the occurrence of a non-selected or unmotivated feature could trigger error recovery.

Head movement reconstruction targets complex heads. A complex head is defined in this work as a semi-complex constituent that has only the right constituent, with left constituent missing. The right constituent contains another head (that might itself contain another, recursively). Thus, a complex head such as *admires* will be represented in this work as [$_T$ ∅ [$_v$ ∅ V]]]. Because [∅ X] will be treated as a primitive constituent, it will have its own label and a set of features. Head movement reconstruction repairs these heads by spreading the heads back into the structure. The triggering feature could be the fact that [∅ X] is illegitimate or uninterpretable structure at LF. If so, we add a fourth input condition: illegitimate constituency structure. That condition is matched with an equivalent output condition, which requires that constituency structure is interpretable at the LF-interface. All in all, there are at least four conditions that trigger error recovery:

(40) *Input conditions*
A. Violation of specifier or complement selection,
B. violation of probe-goal (tail-head),
C. presence of an unselected or unmotivated feature,
D. violation of constituency.

It is clear again from A-D that transfer is reacting to violations and attempts to fix them by using a ‘dropping’ operation. This raises the question of why such violations are part of the language in the first place. It is possible to imagine a “super-analytical language” in which everything occurs in the surface string in a position that allows it to be merged directly to its normalized position, and such language would also be more optimal in terms of the computational resources it consumes. The fact that transfer functions like a hardwired reflex means that speakers can produce (intentionally or accidentally) noncanonical sentences that are normalized in a reflex-like manner. The fact that these noncanonical objects affect semantic interpretation means that they may become functional in language use, which could provide one path for canonicalization.

2.9 Lexicon and morphology

Most phonological words enter the system a polymorphemic units that might be further associated with inflectional features. The morphological component is responsible for decomposing phonological words into these components. A table-look up dictionary matches phonological words directly with their morphological decompositions. This corresponds with an automatized pattern recognition procedure. The decomposition consists of a linear string of morphemes $m_1\#...\#m_n$ that are separated and inserted into the linear input individually (41). Notice the reversed order.

(41)	Nukku-u-ko	+	Pekka	→	Q	+	T/fin	+	V	+	Pekka
	sleep-T/fin-Q		Pekka		↓		↓		↓		↓
	sleep#T/fin#Q				[Q		[T/fin		[V		Pekka]]]
	‘Does Pekka sleep?’										

The primitive morphemes are matched with the lexicon (table-lookup) and retrieve lexical items. Lexical items are provided to the syntax as primitive constituents, with all their properties (features) coming from the lexicon.[6] This is illustrated in Figure 6.

[6] Another morphological system is comprised of generative morphology that is called for when an input word does not match with anything in the table-lookup dictionary. Generative morphology parses unrecognized words and/or guesses their feature composition (e.g., label).

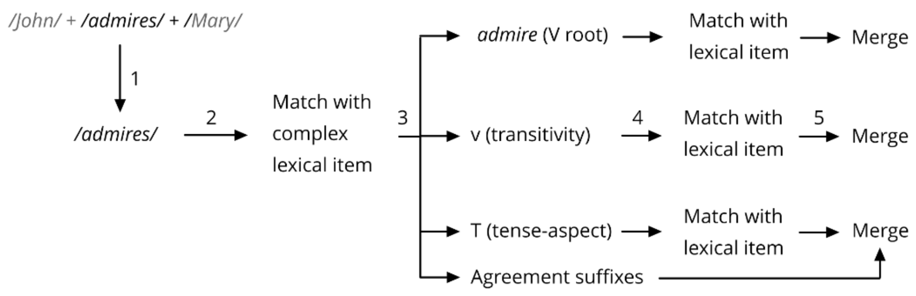

Figure 6. A schematic overview of the structure of the lexicon and the computational operations involved in lexical decomposition. The whole pipeline contains the following steps: (1) consume a phonological word from the input; (2) match the word with a lexical item, or a (ranked) list of lexical items if it is ambiguous (in this study, a disambiguated lexicon was used); if the matched item was complex, broken it down into primitive parts (3) each which is then matched with a primitive lexical item in the same lexicon (4); Merge the word into the phrase structure build so far. The order of Merge is from the end of the word towards to the root (here from bottom to up: T-v-V).

Inflectional features (such as case suffixes) are listed in the lexicon as items that have no morphemic content. They are extracted like morphemes, inserted into the input sequence, but converted into *features* instead of morphemes in syntax.

Lexical features emerge from three distinct sources. One source is the language-specific lexicon, which stores information that is specific to a lexical item in a particular language. For example, the Finnish sentential negation behaves like an auxiliary, agrees in φ-features, and occurs above the finite tense node in Finnish (Holmberg et al. 1993). Its properties differ from the English negation *not.* Some of these properties are so idiosyncratic that they must be part of the language-specific lexicon. One such property could be the fact that the negation selects T as a complement, which must be stated in the language-specific lexicon to prevent the same rule from applying to the English *not*. It is assumed that language-specific features *override* features emerging from the two remaining sources if there is a conflict.

Another source of lexical features comes from a set of universal redundancy rules. For example, the fact that the small verb v selects for V need not be listed separately in connection with each transitive verb. This fact emerges from a list of universal redundancy rules which are stored in the form of feature implications. These redundancy rules make up a 'mini grammar'. In this case, the redundancy rule states that the feature [CAT:v] implies the existence of feature [COMP:V]. Broad verb classes (e.g., transitive, unaccusative) can be defined as (macro)features that are associated with a set of redundancy rules. When a lexical item is retrieved, its feature content is first fetched from the language specific lexicon and is then processed through the redundancy rules and parameters. If there is a conflict, the language-specific lexicon wins. Lexical features constitute an unstructured set, but the features themselves have often a minimal ‘type:value’ structure. For example, labels are provides as values (N, V, A, …) of the type CAT(egory), hence CAT:N, CAT:V, etc.

A third source for lexical features is constituted by what one might initially classify as “parameters.” There are many cases in which a redundancy rule (or something that resembles one) is applied in a particular language or in a group of languages. For example, in Finnish most heads that have the [SPEC:*] feature also exhibit φ-agreement in some context (e.g., prepositions, noun heads), whereas the same functional heads in English have neither (Brattico, Chesi, and Suranyi 2019). These parameters are currently implemented by a separate code that is executed during lexical retrieval. Another hypothesis is that they are based on a system of language-specific lexical redundancy rules; a third would be to assume that they emerge in syntax during the parsing derivation and respond to language-specific features. The correct implementation will ultimately depend on the nature of the rules themselves.

2.10 Argument structure

Argument structure refers to the structure of thematic arguments at their canonical LF-positions, the latter which are defined both by means of theta role assignment and by tail-head dependencies. The parser-grammar will usually have to reconstruct the argument structure from the input string due to several displacement operations.

The thematic role of ‘agent’ is assigned at LF by the small verb v to its specifier. The small verb therefore has feature [!SPEC:D]. A DP argument that occurs at this position will automatically receive the thematic role of ‘agent’. The parser-grammar does not see the interpretation, however, only the selection feature. Thus, when examining the output of the parser, the canonical positioning of the arguments must be checked against native speaker interpretation. The sister of V receives several roles depending on the context. In a v-V structure, it will constitute the ‘patient’. In the case of an intransitive verb, we may want to distinguish left and right sisters: right sister getting the role of patient (unaccusatives), left sister the role of ‘agent’ (unergatives). Their formal difference is such that a phrasal left sister of a primitive head constitutes both a complement and a specifier (per formal definition of ‘specifier’ and ‘complement’), whereas a right sister can only constitute a complement. This means that unaccusatives and unergative verbs can be distinguished by means of lexical selection features: the latter, but not the former, can have an extra specifier selection feature, correlating with the ‘agentive’ interpretation of the argument. Thus, a transitive verb will project three argument positions Spec,vP, Spec,VP and Comp,VP, whereas an intransitive two, Spec,VP and Comp,VP. This means that both constructions have room for one extra (non-DP) argument, which can be filled in by the PP. Ditransitive clauses are built from the transitive template by adding a third (non-DP) argument. They can be selected, e.g. the root verb component V of a ditransitive verb can contains a [!SPEC:P] feature. Ideally, verbal lexical entries should contain a label for a whole verb class, and that feature should be associated with its feature structure by means of lexical redundancy rules.

Adverbial and other adjuncts are associated with the event by means of tail-head relations. A VP-adverbial, for example, must establish a tail-head relation with a V. Adverbial-adjunct PPs behave in the same way. The tail-head relation can involve several features. For example, the Finnish allative case (corresponding to English 'to' or 'for') must be linked with verbs which describe 'directional' events (42). It therefore tails a feature pair CAT:V, SEM:DIRECTIONAL. There is no limit on the number of features that a verb can possess and a prepositional argument that tail.

(42)

a.	*Pekka	näki	Merjalle.
	Pekka	saw	to.Merja
b.	Pekka	huusi	Merjalle.
	Pekka	yelled	at.Merja

The syntactic representation of an argument structure raises nontrivial and interesting questions concerning the relationship between grammar and meaning. Consider a simple sentence such as *John dropped the ball* and a conscious, perceptual or imaginary representation of its meaning, illustrated in Figure 8.

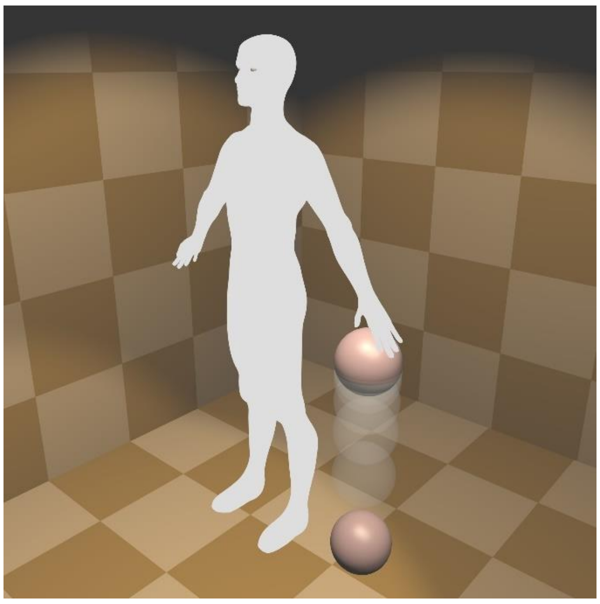

Figure 8. A non-discursive, perceptual or imagined representation of the meaning of the sentence *John dropped the ball.*

The canonical LF-representation for the sentence is mapped compositionally to the above representation in such a way that the innermost V-DP configuration represents the event of ball's falling, while the v-V adds the meaning that the event is originated by an agent or force. The subject then adds what that force is. This provides the idea that it is John that causes the ball to fall or is 'responsible' for the event's occurrence Therefore, addition of elements to the LF-structure adds elements to the event (as imagined, perceived or hallucinated). Some functional heads add independent components, like aspect or tense, others, like v, make up unsaturated components ‘x causes an event’ that require a further object ‘x’. All functional heads require a complement, a given representation/meaning. Under this conception, syntactic selection features allow the language faculty to create configurations that can be provided a legitimate interpretation of the type illustrated in Figure 8. On the other hand, they do not guarantee that an interpretation emerges: a sentence like *John dropped the democracy furiously* is as possible as *John dropped the ball by accident*. Furthermore, idioms cannot always be interpreted literally.

2.11 Agreement, null subjects and control (Agree^{-1})

Inflectional φ-features of predicates, such as verbs, are extracted from the input and embedded as morphosyntactic features to the corresponding lexical items (43).

```
(43) John      admire + s    Mary.
       ↓           ↓           ↓
     [John     [admire       Mary]]
               {3sg}
```

These features are then used to saturate or value the unvalued φ_ features of lexical items. For example, when deriving a sentence such as (44), the lexical constituent *admire* will contain two features relevant for agreement: an unvalued φ-set designated by φ_ (from the lexicon), and the agreement suffixes {3sg} extracted from the input.

```
(44) John      admire+s      Mary
     [John     [admire       Mary]]
               φ_ = 3sg
```

More generally, during Transfer, an *agreement reconstruction* (Agree^{-1}) takes place after all movement has been reconstructed that tries to value φ_ for each head possessing the feature. Agreement reconstruction first tries to value φ_ at head H by locating a local DP-argument from (i) within its sister or (ii) from its SPEC and, if not found, uses the φ-set obtained from the input (if any). The φ-set obtained from the input must project an unambiguous pronoun and cannot involve conflicts; English {3sg} does not satisfy this condition because it is consistent with conflicting gender features. In Finnish and Italian, verbal φ-suffixes are unambiguous and are treated as ‘pronominal’ enough to value φ_, which leads into the familiar pro-drop profile: the subject can remain covert if an unambiguous φ-set is present at the verb. If nothing can value φ_, antecedent recovery is attempted at LF that tries to find a local c-commanding antecedent that could be used to value the features. In the example (45), the verb to leave cannot value its φ_set from an overt argument or from agreement suffices, so the feature are valued at LF by locating the antecedent John. This creates the phenomenon known as “control.”

(45) John wants to leave$_{\varphi\_=\text{John}}$.

If no antecedent is found, a generic or arbitrary interpretation results, in which the argument of a predicate refers to 'people in general' (46).

(46) To leave$_{\varphi\_=\text{arb}}$ now would be a mistake.

The unvalued φ_ expresses the fact that a predicate is unsaturated, in the Fregean sense: it must be linked with a referential object in order to generate an intelligible object. For example, it is impossible to imagine what 'falling' would constitute unless there were some object that falls. The predicate constitutes the unsaturated part – a functional element with φ_ – which is saturated by means of valuation. Agreement and control, then, constitute a reflex of saturation and both constitute a 'repair' strategy.

3 Formalization and implementation

3.1 Introduction

The full formalization of the theory outlined above is provided in the form of a Python implementation. That implementation provides formal definitions for every concept and computational operation. This document discusses them in a less technical manner and provides an empirical justification when it was not deemed self-evident from the context. No actual computer code is cited in this document. Thus, this document should serve as an introduction to the implementation that one should ideally consult before examining the source code. In addition, the parser undergoes continues revisions and additions, while the documentation often lags a few steps behind.

The motivation for formalization requires a comment. It is all too often that we encounter the belief that formalization would constitute the end goal of science. In linguistics this mistaken belief manifests in various attempts at developing formal grammars and formalizations of this or that system. While such formalization might serve some aesthetic purpose, it is not useful unless the formalization is employed in some operation that contributes to the underlining research agenda. Thus, the purpose of formalization in the mature hard sciences is not just to formalize some set of assumptions, but to calculate (demonstrate, prove) with mathematical precisions the predictions of one's theory, and then compare these predicted results with the data gathered from observation. Inferential equations, for example, are used to calculate what the consequences of one's assumptions are; they do not exist for the sole purpose of presenting the assumptions in a precise format. They are precisely what they are *because* they can be used to calculate; that is the whole point. It is important to emphasize the notion of "calculating": it is again very little use to formalize something and then use nothing but ordinary language to derive consequences; why to formalize anything if no formal calculations are performed? Rather, formalization should be done in order to *literally calculate* what the predictions and consequences are. The role of the formal computer program in the present content is to provide the means to calculate what the theory predicts; it corresponds to a particular set of differential equations in a particular physical theory. It is neither a practical parser aimed at parsing everything, nor a formalized grammar that one could adopt or reject for some subjective reason.

Furthermore, there is no reason to not to formalize one's theory and then calculate what the consequences are. Perhaps in the past, with computational resources and the required skills being scarce, there was a legitimate concern that such efforts would hamper progress and divert resources into unnecessary activities (writing and debugging code, maintaining hardware). These concerns a long gone; in today's world, computational resources are plenty and transforming one's research ideas into precise form that can be used to construct rigorous proofs is *trivial*. The parser described here consists of few hundred lines of functional code (ignoring comments, support functions and other parts that are not part of the theory); furthermore, the code is written in a high-level language (Python) that can be understood by anybody. It is inevitable that as theoretical linguistics matures, it will move to use these methods exclusively and in a manner they are used in other natural sciences; no theoretical physicist would dream of doing his or her work without mathematical tools. Progress will be faster: the testing and evaluating cycle will speed up considerably.

3.2 Overall structure of the algorithm

The generative linear phase parser-grammar implements a recursion over a search space defined by the ranking of the merge sites. The basic structure of the recursion is illustrated in Figure 1.

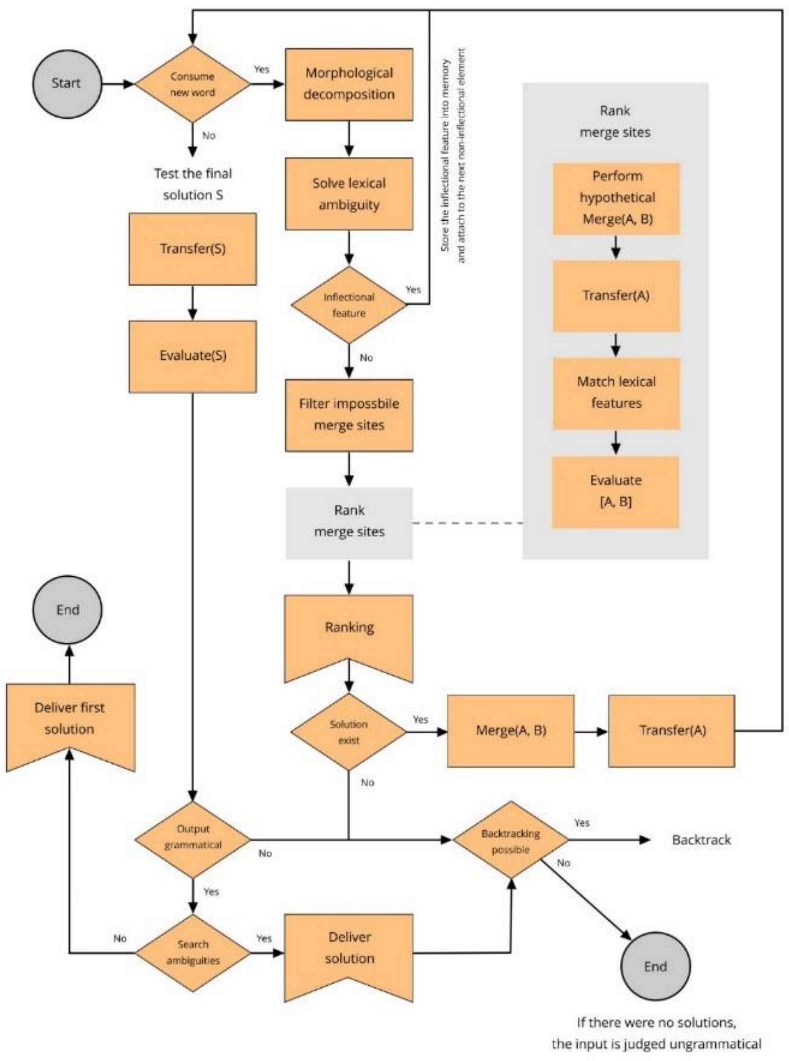

Figure 1. The structure of the generative linear phase parser-grammar language comprehension algorithm.

The individual components illustrated in this Figure are commented in the subsections below.

3.3 The parser

3.3.1 Phillips cycle

The recursive parsing function takes the currently constructed phrase structure α, a linearly ordered list of words and an index in the list of words as its arguments. It will first check if the whole clause has been consumed and, if it is, α will be transferred to LF and evaluated. Transfer normalizes the phrase structure, thus recovers from errors, reverse-engineers movement and agreement and performs LF-legibility tests to check that the output is semantically interpretable. If the test passes, the result will be accepted: the system has "understood" the expression (in one way). Executive control is then passed to the conceptual-intentional system that is not modeled here.

Suppose word w was consumed. To retrieve properties of w, lexicon will be accessed. This operation corresponds to a stage in language comprehension in which an incoming sensory-based item is matched with a lexical entry. If w is ambiguous, all corresponding lexical items will be returned as a list [w_1, w_2, . . ., w_n] that will be explored in some order. The ordered list will be added to the recursive loop as an additional layer. The ordering is arbitrary in the current interpretation but not so in a realistic parsing scenario.

Next, a word from this list, say w_1, will be subjected to morphological parsing. Suppose the word is *pudo-t-i-n-ko-han* 'fall-cau-past-1sg-Q-hAn'. Morphological parser will return a new list of words that contains the individual morphemes that were part of w_1 together with the lexical item corresponding with the first item in the new list. The new list therefore contains a morphological decomposition of the word. Some of the morphemes in word w could be inflectional: they are stored as features into a separate memory buffer and then added to the next morpheme when it is being consumed. If inflectional features were encountered instead of a morpheme, the parsing function is called again immediately without any rank and merge operations. If there were several inflectional affixes, they would be all added to the next morpheme m consumed from the input. A complex phonological word such as *pudo-t-i-n-ko-han* will enter syntax in the form (47). Notice that the order of the morphemes is reversed.

(47)	pudo-t-i-n-ko-han	(input word)
	fall-cau-past-1sg-Q-hAn $\rightarrow$	(decomposition)
	hAn + Q + 1sg + T + v + V	(reverse order into syntax)

Suppose morpheme m, say *hAn*, was consumed. The morpheme must be merged to the existing phrase structure into some position. Merge sites that can be ruled out as impossible by using local information are filtered out. The remaining sites are then ranked (Section 3.3.2). Each site from the ranking is explored. For each site there are two options: if the new morpheme α was part of the same word as the previous morpheme β, it will be embedded into the previous word to create a complex head [$_{\beta}\varnothing$, α]. Thus, the morphemes in (47) are first processed in this way and create a complex head [$_{hAn}\varnothing$, [$_{Q}\varnothing$, [$_{T}\varnothing$, [$_{v}\varnothing$, V]]]]. If α was not inside the same word as the previous item, it will be merged countercyclically to the existing phrase structure, and the parsing function is called recursively. If a ranked list has been exhausted without a legitimate solution, the algorithm will backtrack to an earlier step.

The complex head [$_{hAn}\varnothing$, [$_{Q}\varnothing$, [$_{T}\varnothing$, [$_{v}\varnothing$, V]]]] corresponds to a situation in the more standard bottom up theories in which all of the morphemes have been 'snowballed' out of the structure to the highest node. In the standard theories, the heads would have been adjoined with each other. This solution did not produce elegant results here, because any constituent with both left and right constituents is automatically regarded as a standard complex constituent and not a head. This would make any complex head a phrasal specifier. Various ad hoc strategies are used in the standard theories to prevent this from happening, but these devices are questionable in the parsing context.

3.3.2 Filter and ranking

Filtering. Several conditions can be exploited in filtering potential merge sites so that they do not appear in the rankings and do not consume computational resources. Let us assume that α is the new element to be merged and β is the site of merge.

(48) *Conditions for filtering*

a. Filtering Condition A. If the new morpheme α was inside the same word as the previous one, all other solutions except 'merge to the complement' (bottom of β) will be filtered out. Thus, v can only be complemented to T if they were originally inside the same word in the input. The rationale for this rule is obvious.

b. Filtering Condition B. Primitive sites that do not accept any type of complements are filtered out. Justification is trivial.

c. Filtering Condition C. If β is not primitive, it will be reconstructed (Section 3.4.2) and tested for LF-legibility: the legibility of β is always tested upon [β, α], and if β does not pass the solution is either rejected or ranked lower, depending on the type and severity of the violation. The filter function does not implement ranking, only filtering. Notice that the legibility test for β presupposes that β is normalized and goes through transfer first. Complete rejection occurs if (i) LF-legibility fails, (ii) α is not adjoinable and (iii) the probe-goal test, head integrity test and the criterial feature integrity test (Section 3.8) all failed (if any of them succeeds, filtering is blocked).

Conditions A-B are self-evident, but Condition C requires a comment. The reason [β, α] can be filtered under some conditions is because some legibility errors in β cannot be fixed in the derivation's future. Hence the solution can be rejected. Condition (iii) mean that the failure of the three tests in conjunction is so severe that the problem can be deemed as unrepairable, but it might be that the failure of each test individually is sufficient (the matter requires further examination and is possibly a mistake). The issue impacts computational complexity but not grammaticality.

Ranking. The ranking function weights each solution based on various criteria. If two sites have the same ranking, lower sites in the phrase structure will be prioritized and therefore explored first. This rule affects computational complexity. The ordering does not affect solutions found by the algorithm, only efficiency and the first solution returned by the algorithm. Suppose the new element is α and the site to be tested is β, so that we are testing for [β α]. The criteria used in the current version are the following:

(49) *Ranking criteria*

a. Positive specifier selection (+): β matches for α's specifier/sister selection;

b. Negative specifier selection (–): β matches for α's negative specifier/sister selection;

c. Negative specifier selection of everything (–): α has [–SPEC:*];

d. Infrequent specifier feature (–): β matches for α’s infrequent specifier selection;

e. Do not break existing head dependencies (–): [β α] would break existing β^0-α^0 dependency;

f. Tail-head test (–): check if [β α] would fail α’s tail features;

g. Complement test (+): β (or any morpheme inside β) selects for α as complement;

h. Negative complement test (–): β (or any morpheme inside β) does not select α as complement;

j. Semantic mismatch (–): β and α’s semantic features mismatch;

k. Left branch evaluation (–): Reconstructed β fails LF-legibility;

l. Adverbial tail-head test (–): α has label Adv but fails tail-head test when c-commanded by T/fin;

m. Adverbial tail-head test (+): α has label Adv and satisfies tail-head test when c-commanded by T/fin.

If all solutions are ranked negatively, a geometrical solution will be adopted, according which the largest S will be prioritized that is adjoinable and does not contain T/fin, the latter which means that we do not try to merge above T/fin. One of the most complicated situations is constituted by object relativization, which involves a structure with three consecutive DPs (50).

(50) [$_{\text{DP}}$ A man] [$_{\text{DP}}$ who] [$_{\text{DP}}$ Mary] admires __

The problem with this structure emerges from the fact that there is no local principle for guiding the merge in the case of consecutive DPs. The geometrical heuristic, merging each DP to the top, provides the most efficient solution in most cases, including in the case of (50).

3.4 Definitions for formal primitives

3.4.1 Merge: cyclic and countercyclic

Simple Merge takes two constituents α, β and yields [α, β], α being the left constituent, β the right constituent. Countercyclic Merge^{-1} is a more complex operation. It targets a constituent α inside an existing phrase structure and creates a new constituent γ by merging β either to the left or right of α: γ = [α, β] or [β, α]. Constituent γ then replaces α in the phrase structure, with the phrase structural relations updated accordingly. If α is primitive and an adjunct, γ will be adjunct as well. An inverse of countercyclic Merge^{-1} is *remove*, which removes a constituent from the phrase structure and repairs the hole. This operation is used when the parser attempts to reconstruct movement: it merges elements into candidate positions and removes them if they do not satisfy a given set of criteria.

Both Merge to the right and left, and both countercyclically and by extending the structure, are allowed. The range of options is compensated by the restricted conditions under which each operation is allowed.

3.4.2 Sister, specifier, complement

Sisterhood. Iterative application of merge creates a structure that is used to define basic grammatical relations. A *sister* of α is the non-α daughter of the first branching node from α that is not right-adjunct. In addition, right adjuncts don't have sisters, so that they are "isolated." It follows that in [α, β] both α and β are each other's sisters. In [⟨α⟩, β] the same holds, but in the mirror case [α [β, ⟨γ⟩]] the sister of β is α. Right adjuncts are invisible for sisterhood. A separate relation of *geometrical sisterhood* takes them into account.

Specifier. The notion of specifier has undergone an evolution and will be further simplified in the future. One novel aspect of the current model is the hypothesis that a 'pronominal element' (φ-set) inside a head is part of its specifier, hence the notion of specifier is identical with that of the *edge*.

Local specifier. We define the notion of *local specifier* so that it constitutes the non-adjunct left phrasal sister of the node immediately dominating α. Furthermore, if no such local left phrase exists inside αP, then the φ-features inside α, if any, can constitute a 'pronominal specifier' if the φ-set allows for the construction of an unambiguous pronoun (or clitic). This means that the notion of "local specifier" covers local non-adjunct left phrases inside αP plus pronominal elements inside α. The relevance of local specifier is that it is the notion that enters into specifier selection.

(51) Local specifier

The local specifier for a left head α is β such that

a. $[_{\alpha P}$ β [α XP], β is complex (non-head) and β is not adjunct or, if no such element exist,

b. a pro-element (if any) extracted from α.

A "pro-element" is an unambiguous φ-set associated with D (clitic if [D N]). Notice that the definition of local specifier is very close to the definition of "edge," and will be so defined in the future. The specifier of a right head α is its complex left sister β (i.e. β in [β, α]), whether an adjunct or not. This condition is written as a separate condition inside the definition of specifier for clarity, but a uniform generalized notion is of course possible.

A more general definition of a *generalized specifier* is provided, which denotes any left phrase inside αP together with any pronominal material inside α if nothing else is found. Generalized specifier can be a left adjunct. Thus, the set of generalized specifiers of α contains all left phrases inside αP plus the pronominal element inside α, if any.

(52) Generalized specifier

β is a generalized specifier for head α if and only if either (a.) or (b.)

a. β occurs in a configuration $[_{\alpha P}$ β αP] and is a non-head (adjunct or not);

b. β is extracted from α as a pro-element when nothing satisfies (a).

In the future implementation, the notion of local specifier and generalized specifier will be replaced with the notion of *edge*. The definition is part of the current implementation but not fully utilized. An edge contains all left phrases of αP plus the pronominal element inside α, which will be returned in the order of locality. The operation that utilize specifiers and the edge will then be modified so that they look only at the relevant objects (e.g., local specifier only) inside the edge.

(53) Edge of α

The edge of α consists in an ordered list of constituents such that

a. the first element is the pronominal element inside α, if any;

b. then any complex β such as $[_{\alpha P}\ \beta\ \alpha P]$, in the order of locality from α.

The edge thus contains α and its left phrases within its projection (second merge objects).

A *complement* of α is its *right sister*. LF-legibility uses a broader criterion, in which the complement selection feature is checked by sisterhood. The latter assumption (complement = sister) is the more correct one, thus it is possible that a later iteration of the model should abandon the former and use only the latter.

3.4.3 Downstream, upstream, left branch and right branch

Several operations require that the phrase structure is explored in a pre-determined order. Movement reconstruction, for example, may explore the phrase-structure in a determined order in order to find reconstruction sites. In a typical scenario, the right edge is explored: left branches are phases and not revisited (thus, movement also from within left branches is illicit). The operation of right edge exploration would be trivial to define were there no right-adjuncts, but because right adjuncts are possible, the definition of "right edge" requires clarification. The clarification is that right-adjuncts are ignored when we explore the right edge (for other purposes than merging new words to the phrase structure, see below). Thus, the right edge of [α, β] is β, but the right edge of [α, ⟨β⟩] is α. The intuitive motivation for this definition is that β is inside another syntactic working space. A *downstream* or *downward* relation follows this definition. The *left branch* of [α, β] is α (in the case of [α, ⟨β⟩] the notion of left branch and downward branch coincide). These definitions mean that for any given constituent there are three possible "directions": to go downstream (thus, follow labeling and selection), to turn right (to enter into a right adjunct), or to turn left (enter the left branch). The downward/downstream relations will be used in all movement reconstruction: thus, movement reconstruction into left branches or right branches will not occur, which are the exact correct empirical properties.

Consider sentence (54).

(54) John	+	travelled	+	by	+	using	+	. . .
↓		↓		↓		↓		
[John	[	travelled	[	by		using		

Let’s make the untrivial assumption that the parser-grammar has already at this point determined that it is building an adverbial-adjunct *by using*... If so, then the above definition of “right edge” would require all incoming words to be merged as if the adjunct-adverbial were not present at all. The adverbial would be invisible and not part of the right edge. This gives the wrong result. The next word(s), say *his car*, must be merged inside the adverbial. On the other hand, it is not clear that the parser grammar can determine at this stage that it *is* constructing an adverbial. Another possibility is that it is constructing a complement or a PP argument for the main verb. Therefore, the phrase *by using* is part of the right edge insofar as it has not been promoted into an adjunct status, which is the case here. In general, adjunct promotion need not occur during the first pass parse, but only during transfer.

3.4.4 Probe-goal

The probe-goal relation is used in the standard theory to check the existence of a feature (Chomsky 2000, 2001, 2008). It differs from complement specifier selection features in that whereas complement selection is local, probe-goal is not: the probe-goal is like long-distance complement selection. It is used in this way in the present implementation. Consider the grammatical law which says that a finite C must always select for a finite T. We could define this as a simple selection feature, but this runs into trouble in Finnish in which the sentential negation head can intervene between C and T.

(55) C	Pekka	ei	nuku.
C	Pekka	Neg	T

‘Pekka does not sleep.’

The problem cannot be solved by assuming Finnish-specific selection C→Neg and Neg→T, because the negation is not obligatory, and adding an optional rule C→T would still allow the parser to create illegitimate C-H configurations, H being any head. We need a rule that forces the C→T connection, but if we posit it as obligatory complement selection, then (55) becomes ungrammatical because here C selects for Neg. The solution to this problem is obligatory C→T selection that is not local. This is achieved by associating C with lexical feature (!PROBE:CAT:T/FIN). The existence of the probe-feature triggers search for the *goal feature*, here (CAT:T/FIN). Suppose P is the probe head, G is the goal feature, and α is its (non-adjunct) sister in configuration [P, α]; then:

(56) Probe-goal

Under [P, α], G the goal feature, search for G from left constituents by going downwards inside α along its right edge (thus, ignoring right adjuncts and left branches).

For everything else than criterial features, G must be found from a primitive head to the left of the right edge node. Thus, if [H, α] is at the right edge and H is a primitive constituent, feature G is searched from H. If H is complex, it will not be explored (unless G is a criterial feature). The fact that criterial feature search must be separate from the search of other types of features suggest that we are missing some piece of the whole puzzle.

Probe-goal dependencies are subject to limitations that have not been explored fully. The present implementation has an intervention clause which blocks further search if a primitive constituent is encountered at left that has the same label as the probe, but the matter must be explored in a detailed study. Consider again the case of C$\rightarrow$T. When C searches for T, it cannot satisfy the probe-feature by going through another (embedded) C. If it did, lower T would be paired with two C-heads; this is semantically gibberish. Therefore, there is a "functional motivation" for the intervention condition.

3.4.5 Tail-head relation

A tail-head relation is triggered by a lexical feature (TAIL: F_1, … , F_N), F… being the feature or set of features that is being searched from a head. In order for F to be visible for the constituent containing the tail-feature, say T, F must occur in a primitive left head H at the upward path from T. An upward path is the path that follows the mother-of relation. In the example (57), the tail-feature (TAIL:F) at T^0 can see the feature at the head A^0, and B^0, but not at X^0.

(57) [$_{AP}$ A^0 [$_{BP}$[$_{XP}$YP X^0] [B^0 [T^0 ZP]]]]
{F} $\longleftarrow$ {TAIL:F}

For the tail-head relation to be satisfied, all tail-features (if there are several) must be satisfied by the one and the same head. Partial feature match results in failure (and termination of search). Thus, if T has a tail feature (TAIL:F,G), but A has the feature F without G, the tail-head relation fails.

There is certain ambiguity in how the results of a tail-head relation are interpreted. One interpretation is that if full match is not found, the dependency fails. Another is that only the existence of partial match results in a failure; if nothing is matched, the test is still a success. The latter tests if an element is in a "wrong place," the former if it is in the "right place." Both type of tests are useful but in slightly different contexts. The former test is called *external tail-head test,* the latter *internal tail-head test.* The former (external test) is used when checking if a constituent with tail-head features must be moved to another position, in which it would satisfy the test. The latter (internal test) is applied when fitting a constituent with a case suffix and examining if it would appear under a wrong case assigner in that position; here only the presence of a wrong case assigners (tail-head feature) results in a failure, we don't care if nothing is matched. I think there is a better solution.

3.4.6 Minimal finite edge

The *minimal finite edge* (in relation to a constituent α) is the closest node X upstream from α (thus containing α) such as (i) X is inside a projection from label FIN and (ii) X's sister is not. For example, in a configuration C-FinP, the minimal finite edge is FinP. The same procedure could be generalized so that it finds the edge of any projection.

3.5 Movement reconstruction

3.5.1 Introduction

Movement reconstruction is a 'reflex-like' operation that is applied to a phrase structure α and takes place without interruption from the beginning to the end. From an external point of view, it constitutes 'one' step; internally the operation consists of a determined sequence of steps. All movement inside α is implemented if and only if Merge^{-1}(α, β)(Section 2.4). The operation targets α and follows a predetermined order: head movement reconstruction → adjunct movement reconstruction → A´/A-movement reconstruction → agreement reconstruction (i.e. Merge^{-1} → Move^{-1} → Agree^{-1}).

3.5.2 Head movement reconstruction

Head movement reconstruction of $[[_{\alpha}\varnothing, \beta], \gamma]$ can take place in one of two ways:

(58) Head movement reconstruction of $[[_{\alpha}\varnothing, \beta], \gamma]$ can follow (a.) or (b.)

a. $[[\alpha, \beta], \gamma]$,

b. $[\alpha\ [_{\gamma}\ldots\beta\ldots]]$.

Option (a) will be selected if α is primitive, has an affix, and [α, β] satisfies LF-legibility (Section 3.8). In that case no further reconstruction operation will be applied to α. Otherwise option (b) is selected.

The operation travels downward on the right edge of α, targets primitive constituents β at the left that have an affix φ $[_{\beta}\varnothing\ \varphi]$ and drops the head/affix φ downstream if a suitable position is found. Dropping is implemented by fitting the head to the left of each node at the right edge. The position is accepted as soon as one of these conditions is met: (i) the head φ has no EPP feature and can be selected in that position by a higher head; (ii) it has an EPP feature, has a local specifier, and can be selected in that position by a higher head. The left complex sister constitutes an acceptable specifier [XP, φ]. Heads are therefore reconstructed "as soon as a potential position is found."

There are two exceptional conditions. One is if the bottom of the phrase structure has been reached by a head β that has EPP and is selected properly by α, i.e. [α β]. If β has an EPP feature, is selected property, has no specifier, but occurs in configuration [α, β], β being a primitive head without affixes, the solution is accepted without a specifier. The motivation is that the specifier could be filled in later by movement reconstruction.

The configuration is also accepted if the next left constituent downstream is primitive and cannot constitute a specifier (but selection applies). In the latter two cases, the expression can be saved later if the SPEC position is filled in by phrasal movement reconstruction; if not, the derivation will eventually crash. Head φ can itself be complex. The algorithm will encounter the complex [$_{\varphi}$ ∅ γ] later when it travels downwards from the position it originally extracted φ, and applies the same algorithm to γ.

If head reconstruction reaches the bottom, it will try to reconstruct the head to the complement of the bottom node. If no legitimate position was found but the search reaches the bottom, then the head will be reconstructed locally to [α [φ XP]] as a last resort; an unreconstructed head crashes at LF-legibility.

3.5.3 Adjunct reconstruction

Adjunct reconstruction begins from the top of the phrase structure α and targets floater phrases at the left and to the right (e.g., DP at the bottom). If a floater is detected, it will be dropped. A floater has the following necessary properties:

(59) *A definition of a floater*
XP is a *floater* if and only if
(i) it is complex;
(ii) it has not been floated already;
(iii) it has a tail set;
(iv) floating is not prevented by feature [–FLOAT];
(v) external tail-head test fails.

Condition (iv) prevents genitive arguments from floating in Finnish, but this may also apply to English accusative pronouns. An alternative is to exclude these arguments from the tail-head mechanism, but then their canonical position must be retrieved without the tail features. This is another possibility that applies in some cases (genitive arguments) but fails in other (e.g. genitives inside deverbal nominals). Suppose α satisfies conditions (i-iv) above. Then if the tail-head features (Section 3.4.5) are not satisfied (Section 3.5.4), α must be subject to adjunct reconstruction and is designated as a floater.

In addition, (a) if its tail-head features are satisfied but α constitutes a specifier of a finite head with an EPP feature, or (b) if it constitutes a specifier for a head that does not accept specifiers at all [–SPEC:*], it will also be targeted for reconstruction. Condition (b) is self-evident: the position is still wrong. Condition (a) is required when the adverbial and/or another type of floater occurs in the specifier of a finite head where its tail features are (wrongly) satisfied by something in the *selecting* clause. The EPP-feature indicates that the adverbial/floater must reconstruct into its own clause, and not to remain in the high specifier position. In both cases, it is judged to be in a "wrong" position.

After a floater is detected, it will be dropped. Dropping is implemented by first locating the closest finite tense node (Section 3.4.6). Starting from that and moving downstream, the floater is fitted into each possible position. Adverbials and PPs are fitted to the right, everything else to left. Fitting involves three conditions:

(60) *Fitting a floater*

α can be fitted into position P if and only if

(i) tail-head features are satisfied (Section 3.5.4),

(ii) P is not the same position where α was found,

(iii) P is not a local specifier position.

Adverbials and PPs will be merged to right, they have to observe only (i); everything else to left, and by (i-iii). The floater is promoted into an adjunct once a suitable position is found. This will allow it to be treated correctly by selection, labeling and so on.

3.5.4 External tail-head test

External tail-head is defined in the following way.

(61) A head α's tail features are checked if either (a.) or (b.).

a. The tail-features are checked by a c-commanding head;

b. α is located inside a projection of a head having the tail features.

Tail features are checked in sets: if a c-commanding (a) or containing (b) head checks all features of such a set, the result of the test is positive. If matching is only partial, negative. If the tail-features are not matched by anything, the test result is also negative. Thus, the test ensures that constituents that are "linked" at LF with a head of certain type, as determined by the tail features, can be so linked. All c-commanding heads are analyzed; this implements the feature vector system of (Salo 2003). There are several reasons why checking must be nonlocal, long-distance structural case assignment and checking of negative polarity items being a few.

3.5.5 A´/A-reconstruction (Move^{-1})

Suppose A´/A-reconstruction (Move^{-1}) is applied to α. The operation begins from the top of α and searches downstream for primitive heads at the left or (bottom) right. Three conditions separate are checked, each leading into different action:

(62) *The three conditions of A/A-bar reconstruction*

(i) If the head α lacks a specifier it ought to have on the basis of its lexical features (e.g. v), memory buffer is searched for a suitable constituent and, if found, is merged to the SPEC position;

(ii) If the head α has the EPP property and has a specifier or several, they are stored into the memory buffer;

(iii) If the head α misses a complement that it ought to have on the basis of its lexical features, the memory buffer is consulted and, if one is found, it will be merged to the complement position.

The phrase structure is explored, one head at a time, checking all three conditions for each head.

Option (i): fill in the SPEC position. If α does not have specifiers, a constituent is selected from the memory buffer if and only if either (1) α has a matching specifier selection feature (e.g., v selects for DP-specifier) or (2) α is an EPP head that requires the presence of phi-features in its SPEC that can be satisfied by merging a DP-constituent from the memory buffer (successive-cyclicity). Option (2) is not yet fully implemented, as the generalized EPP mechanism involved (Brattico 2016; Brattico et al. 2019) is not formalized explicitly. An additional possibility is if an existing specifier of α is an adjunct: then an argument can be tucked in between the adjunct and the head, where it becomes a specifier, if conditions (i-ii) apply.

Option (ii): store specifier(s) into memory. This operation is more complex because it is responsible for the generation of new heads if called for by the occurrence of extra specifiers. The operation takes place if and only if α is an EPP head: thematic constituents are not targeted. Let us examine the single specifier case first. A specifier refers to a complex left aunt constituent βP such that [βP [α_{EPP} XP]] and βP has not been moved already. If α has the generalized EPP feature, βP will undergo A-reconstruction (local successive-cyclicity, Section 3.5.6); otherwise it will be put into memory buffer. The latter option leads possibly into long-distance reconstruction (A´-reconstruction).

If βP has criterial features (which are scanned from it), formal copies of these features are stored to α. A formal copy of feature F is denoted by uF. If h is a finite head with feature FIN, a scope marker feature iF is created. This means that if F is the original criterial feature, then uF is a formal trigger of movement and iF is the semantically interpretable scope marker/operator feature. Lexical redundancy rules and parameters are applied to α to create a proper lexical item in the language being parsed (α might have language-specific properties). In addition, the label of α will be also copied to the new head, implementing the "inverse of feature inheritance." If α has a tail feature set, an adjunct will be created. This is required when a relative pronoun creates a relative operator, transforming the resulting phrase into an adjunct.

If an extra specifier is found, the procedure is different. If the previous or current specifier is an adjunct and the correct specifier has no criterial features, then nothing is done: no intervening heads need to be projected. If there are two non-adjuncts, a head must be generated between the two, its properties copied from the criterial features of higher phrase and from the label of the head h. The latter takes care of the requirement that when C is created from finite T, C will also has the label FIN. If the new head has a tail feature set, an adjunct is created. This operation is required when a relative pronoun creates a relative operator, transforming the resulting phrase into a relative clause adjunct.

Option (iii): fill in the complement position from memory. A complement for head α is merged to Comp,αP from the memory buffer if and only if (1) α is a primitive head, (ii) α does not have a complement or α has a complement that does not match with its complement selection, and (iii) α has a complement selection feature that matches with the label of a constituent in the memory buffer.

Once the whole phrase structure has been explored, extraposition operation will be tried as a last resort if the resulting structure still does not pass LF-legibility (Section 3.5.7).

3.5.6 A-reconstruction

A-reconstruction is an operation in which a DP makes a local spec-to-spec movement, i.e. [DP_1 [α $\_\_{}_1$ β]]. The operation is implemented if and only if α has the generalized EPP property: we assume that DP has been moved locally to satisfy this feature.

3.5.7 Extraposition as a last resort

Extraposition is a last resort operation that will be applied to a left branch α if and only if after reconstruction all movement α still does not pass LF-legibility. The operation checks if the structure α could be saved by assuming that its right-most/bottom constituent is an adjunct instead of complement. This possibility is based on ambiguity: a head and a phrase 'k + hp' in the input string could correspond to [K HP] or [K ⟨HP⟩]. Extraposition will be tried if and only if (i) the whole phrase structure (that was reconstructed) does not pass LF-legibility test and (ii) the structure contains either finiteness feature or is a DP. Condition (i) is trivial, but (ii) restricts the operation into certain contexts and is nontrivial and possible must be revised when this operation is examined more closely. A fully general solution that applied this strategy to any left branch ran into problems. If both tests are passed, then the operation finds the bottom HP = [H XP] such that (i) HP is adjoinable in principle (Brattico 2012) and either (i.a) there is a head K such that [K HP] and K does not select HP or K obligatorily selects something else (thus, HP *should* be interpreted as an adjunct) or (i.b) there is a phrase KP such as [KP HP].[7] HP is targeted for possible extraposition operation. Next, it will be checked (redundantly?) that H is c-commanded by FIN or D and, if it is, HP will be promoted into adjunct (Section 3.5.8). This will transform [K HP] or [KP HP] into [K ⟨HP⟩] or [KP ⟨HP⟩], respectively. Only the most bottom constituent that satisfies these conditions will be promoted; if this does not work, and α is still broken, the model assumes that α cannot be fixed.

[7] The notion "is adjoinable" means that it can occur without being selected by a head. Thus, VP is not adjoinable because it must be selected by v.

To see what the operation does, consider the input string *John gave the book to Mary*. Merging the constituent one-by-one without extraposition could create the following phrase structure, simplifying for the sake of the example:

(63)	*John*	*gave*		*the book*	*to*	*Mary*
	[John	[T	[$_{\text{DP}}$[	the book	[P	Mary]]]
				SPEC	P	COMP

This interpretation is wrong. First, it contains a preposition phrase [*the book* P DP], which is ungrammatical in English (by most analyses). Second, the verb *gave* now has the wrong complement PP when it required a DP. Extraposition will be tried as a last resort, in which it is assumed that the string 'D + N + [P + D + N]' should be interpreted as [DP ⟨HP⟩]. This will fix both problems: the verb now takes the DP as its complement (recall that the right adjunct is invisible for selection and sisterhood), and the preposition phrase does not have a DP-specifier. It is easy to see how the PP satisfies the criteria for the application of the extraposition operation: PPs are adjoinable, the configuration is [DP PP], and the PP is inside a FinP. The final configuration that passes LF-legibility test is (64).

(64) [John [gave [$_{\text{DP}}$[the book] ⟨to Mary⟩]]]

There is one more detail that requires comment: the labeling algorithm will label the constituent [DP ⟨PP⟩] as a DP, making it look like the PP adjunct were inside the DP. This is not the case: it is only attached to the DP geometrically, but is assumed to be inside the secondary syntactic working space. No selection rule "sees" it inside the DP. On the other hand, if this were deemed wrong, then the operation could attach the promoted adjunct into a higher position. This would still be consistent with the input '*the book to Mary*'.

3.5.8 Adjunct promotion

Adjunct promotion is an operation that changes the status of a phrase structure from non-adjunct into an adjunct, thus it transfers the constituent from the "primary working space" into the "secondary working space." Decisions concerning what will be an adjunct and non-adjunct cannot be made in tandem with consuming words from the input. The operation is part of reconstruction. If the phrase targeted by the operation is complex, the operation is trivial: the whole phrase structure is moved. If the targeted item is a primitive head, then there is an extra concern: how much of the surrounding structure should come with the head? Is it possible to promote a head to an adjunct while leaving its complement and specifier behind? It is obvious that the complement cannot be left behind. If H is targeted for promotion in a configuration [$_{\text{HP}}$ H, XP], then the whole HP will be promoted. This feature inheritance is part of the adjunct promotion operation itself.

The question of whether the specifier must also be moved is less trivial, and the model is currently unstable with respect to this issue, having undergone several revisions. The matter must be examined in detail later.

3.6 Agreement reconstruction (Agree^{-1})

Agreement reconstruction (Agree^{-1}) is attempted after all movement has been reconstructed. The operation goes downstream from α and targets any primitive head to the left that has unvalued φ-set (φ_) and requires valuation (does not have feature -VAL). That triggers Agree^{-1}. Such heads H attempt to *acquire* φ-features. Acquisition of φ-features consists of two steps: (i) acquisition from φ-features from within the sister of H and, if unvalued features remain, (ii) acquisition from specifier. Operation (i) triggers phase-bounded downward search for left phrases with label D and collects all valued φ-features from the first such element. Operation (ii) picks up the first generalized specifier (adjunct or non-adjunct) with label D and obtains φ-features from it. Notice that the definition of 'generalized specifier' includes the pro-element, if present, at H itself. That element is only consulted if no phrasal specifier exists; the definition of generalized specifier is such that only if there is no phrase at SPEC will the pro-element be included.

Acquired φ-featured are then *valued* to the head. Denote a φ-feature of the type T with value V as [PHI:T:V]. An acquired φ-feature {PHI:T:v} can only be valued at H if (i) H contains {PHI:T:_} and (ii) no conflicting feature {PHI:T:v´} exists at H. Condition (ii) leads into *φ-feature conflict* which, when present, crashes the derivation at LF. Thus, condition (ii) does not terminate the operation, but is illegitimate at LF.

3.7 Morphological and lexical processing

The parser-grammar reads an input that constitutes a one-dimensional linear string of elements at the PF-interface that are assumed to arise through some sensory mechanism (gesture, sound, vision). Each element is separated from the rest by a "word boundary." A word boundary is represented by space, although no literal 'space' has to exist at the sensory level. Each such element at the PF-interface is then matched with items in the lexicon, which is a repository of a pairing between elements at the PF-interface and lexical items.

A lexical element can be *simple* or *complex*. A complex lexical element consists of several further elements separated by a #-boundary distinguishing them from each other inside phonological words. This correspond to a "morphologically complex word." A morphologically complex word cannot be merged to the phrase structure as such. It will be decomposed into its constituent parts, which will be matched again with the lexicon, until simple lexical elements are detected. A simple lexical element corresponds to a *primitive lexical item* that can be merged to the phrase structure and has features associated with it. For example, a tensed transitive finite verb such as *admires* will be decomposed into three parts, T/fin, v and V, each which is matched with a primitive lexical item and then merged to the phrase structure. Morphologically complex words and simple words exist in the same lexicon. A decomposition can be given also directly in the input. For example, applying prosodic stress to a word is equivalent to attaching it with a #foc feature. It is assumed that the PF-interface that receives and preprocesses the sensory input is able to recognize and interpret such features. The morphological parser will then extract the #foc feature at the PF-interface and feed it to the narrow syntax, where it becomes a feature of a grammatical head read next from the input.

It is interesting to observe that the ordering of morphemes within a word mirrors their ordering in the phrase structure. The morphological parser will therefore reverse the order of morphemes and features inside a phonological word before feeding them one by one to the parser-grammar. Thus, a word such as /admires/ → admire#v#T/fin will be fed to the parser-grammar as T/fin + v + admire(V). The same effect could be achieved by other means.

There are three distinct lexical components. One component is the language-specific lexicon (lexicon.txt) which provides the lexical items associated with any given language. Each word in this lexicon is associated with a feature which tells which language it belongs to. The second component hosts a list of universal redundancy rules which add features to lexical items on the basis of their category. In this way, we do not need to list in connection with each transitive verb that it must take a DP-complement; this information is added by the redundancy rules. The redundancy rules constitute in essence a 'mini grammar' which tell how labels and features are related to each other. The third component is a set of *universal lexical items* such as T, v, Case features, and many others. When a lexical element is created during the parsing process, for example a C(wh), it must be processed through all these layers, while language must be assumed or guessed based on the surrounding context.

A lexical item is an element that is associated with a *set of features* that has also the property that it can be merged to the phrase structure. In addition to various selection features, they are associated with the label/lexical category (CAT:F), often several; phonological features (PF:F); a semantic *concept* interpretable at the LF-interface and beyond (LF:F)(of the type delineated by Jerry Fodor 1998); topological semantic field features (SEM:F); language features (LANG:F), tail-head features (TAIL:F, …G), probe-features (PROBE:F), φ-features (e.g., PHI:NUM:SG) and others. The number and type of lexical features is not restricted by the model.

3.8 LF-legibility

The purpose of the LF-legibility test is to check that the syntactic representation satisfies the LF-interface conditions and can be interpreted semantically. Only primitive heads will be checked. The LF-legibility test consists of several independent tests. It constitutes an internal "perceptual mechanism" that ensures that what is being generates makes sense linguistically.

Suppose we test head H. The *head integrity test checks* that H has a label. A head without label will be uninterpretable, hence it will not be accepted. A *probe-goal test* checks that a lexical probe-feature, if any, can be checked by a goal. Probe-goal dependencies are, in essence, nonlocal selection dependencies that are required for semantic interpretation (e.g. C/fin will select for T/fin over intervening Neg in Finnish). An *internal tail test* checks that D can check its case feature, if any. The *double specifier test* will check that the head is associated with no more than one (nonadjunct) specifier. The *semantic match test* will check that the head and its complement do not mismatch in semantic features. *Selection tests* will check that the lexical selection features of H are satisfied. This concerns all lexical selection features that state mandatory conditions (an adjunct can satisfy [!SPEC:L] feature). *Criterial feature legibility test* checks that every DP that contains a relative pronoun also contains T/FIN. *Projection principle test* checks that argument (non-adjunct) DPs are not in non-thematic positions at LF. Discourse/pragmatic test provides a penalty for multiple specifiers (including adjuncts).

References

Baker, Mark. 1996. *The Polysynthesis Parameter*. Oxford: Oxford University Press.

Brattico, Pauli. 2012. "Pied-Piping Domains and Adjunction Coincide in Finnish." *Nordic Journal of Linguistics* 35:71–89.

Brattico, Pauli. 2016. "Is Finnish Topic Prominent?" *Acta LInguistica Hungarica* 63:299–330.

Brattico, Pauli. 2018. *Word Order and Adjunction in Finnish*. Aarhus: Aguila & Celik.

Brattico, Pauli. 2019. "Finnish Word Order and Morphosyntax." *Manuscript*.

Brattico, Pauli and Cristiano Chesi. 2019. "A Top-down, Parser-Friendly Approach to Pied-Piping and Operator Movement." *Lingua*.

Brattico, Pauli, Cristiano Chesi, and Balasz Suranyi. 2019. "EPP, Agree and Secondary Wh-Movement." *Manuscript*.

Chesi, C. 2012. *Competence and Computation: Toward a Processing Friendly Minimalist Grammar*. Padova: Unipress.

Chesi, Cristiano. 2004. "Phases and Cartography in Linguistic Computation: Toward a Cognitively Motivated Computational Model of Linguistic Competence." Universita di Siena, Siena.

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

Chomsky, Noam. 1992. "Bare Phrase Structure." Pp. 393–439 in *Government and Binding Theory and the Minimalist Program,* edited by G. Webelhuth. Malden, USA: Blackwell.

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

Chomsky, Noam. 2000. "Minimalist Inquiries: The Framework." Pp. 89–156 in *Step by Step: Essays on*

Minimalist Syntax in Honor of Howard Lasnik, edited by R. Martin, D. Michaels, and J. Uriagereka. Cambridge, MA.: MIT Press.

Chomsky, Noam. 2001. “Derivation by Phase.” Pp. 1–37 in *Ken Hale: A Life in Language,* edited by M. Kenstowicz. Cambridge, MA.: MIT Press.

Chomsky, Noam. 2005. “Three Factors in Language Design.” *Linguistic Inquiry* 36:1–22.

Chomsky, Noam. 2008. “On Phases.” Pp. 133–66 in *Foundational Issues in Linguistic Theory: Essays in Honor of Jean-Roger Vergnaud,* edited by C. Otero, R. Freidin, and M.-L. Zubizarreta. Cambridge, MA.: MIT Press.

Ernst, Thomas. 2001. *The Syntax of Adjuncts.* Cambridge: Cambridge University Press.

Holmberg, Anders, Urpo Nikanne, Irmeli Oraviita, Hannu Reime, and Trond Trosterud. 1993. “The Structure of INFL and the Finite Clause in Finnish.” Pp. 177–206 in *Case and other functional categories in Finnish syntax,* edited by A. Holmberg and U. Nikanne. Mouton de Gruyter.

Jelinek, Eloise. 1984. “Empty Categories, Case and Configurationality.” *Natural Language & Linguistic Theory* 2:39–76.

Phillips, Colin. 1996. “Order and Structure.” Cambridge, MA.

Phillips, Colin. 2003. “Linear Order and Constituency.” *Linguistic Inquiry* 34:37–90.

Salo, Pauli. 2003. “Causative and the Empty Lexicon: A Minimalist Perspective.” University of Helsinki.