

**A computational implementation of a generative linear phase parser.  
Framework and technical documentation**

Version 2.x

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 parser-grammar. The parser-grammar assumes that the core computational operations of narrow syntax (e.g., Merge/Move/Agree) are applied incrementally and 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).<sup>1</sup>

---

<sup>1</sup>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. This document describes the third iteration of the program. The first version (late 2018) implemented operator movement in a way that took pied-piping and internal wh-movement into account; the second version (Spring 2019) then added and tested the full tail-head mechanism to process free word order, while this version adds processing of agreement, pro-drop and control. Changes from one version into another are not necessarily cumulative. The source code is available at <https://github.com/pajubrat/parser-grammar>, in which various previous versions exist as branches. Because the code is used as a research tool, iteration from one version into another follows the structure of empirical research conducted in the background instead of practical use or pure coding criteria. This document will be updated together with each major revision.

## Table of Contents

1	Introduction .....	4
2	The framework .....	5
2.1	Introduction .....	5
2.2	Merge.....	9
2.3	The lexicon and lexical features .....	13
2.4	Phases and left branches.....	16
2.5	Labeling.....	17
2.6	Adjunct attachment.....	19
2.7	EPP .....	23
2.8	Move.....	23
2.8.1	Movement and error tolerance.....	23
2.8.2	General properties of Transfer.....	26
2.8.3	A-bar reconstruction.....	27
2.8.4	A-reconstruction and EPP .....	29
2.8.5	Head reconstruction.....	29
2.8.6	Adjunct reconstruction .....	30
2.9	Ordering of operations.....	31
2.10	Lexicon and morphology.....	32
2.11	Argument structure.....	34
2.12	Agreement, null subjects and control .....	36
3	Formalization and implementation.....	37
3.1	Overall structure of the algorithm .....	37
3.2	The parser .....	38
3.2.1	Phillips cycle .....	38
3.2.2	Filter and ranking .....	38
3.3	Definitions for formal primitives.....	40
3.3.1	Merge: cyclic and countercyclic.....	40

3.3.2	Sister, specifier, complement.....	40
3.3.3	Downstream, upstream, left branch and right branch.....	41
3.3.4	Probe-goal.....	42
3.3.5	Tail-head relation.....	43
3.3.6	Minimal finite edge .....	43
3.4	Movement reconstruction .....	43
3.4.1	Introduction .....	43
3.4.2	Head movement reconstruction .....	44
3.4.3	Adjunct reconstruction .....	44
3.4.4	External tail-head test .....	45
3.4.5	A'/A-reconstruction.....	45
3.4.6	A-reconstruction .....	47
3.4.7	Extraposition as a last resort.....	47
3.4.8	Adjunct promotion.....	48
3.5	Agreement and control .....	49
3.6	Morphological and lexical processing.....	49
3.7	LF-legibility.....	51

## 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 that knowledge in use in real-time communication. What kind of mistakes do 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 in 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 the parser, a computational system that generates (correct) syntactic representations for linguistic input available to the hearer. Whether language comprehension/parsing is or is not part of the theory of linguistic competence is an empirical question that cannot be settled by conceptual or methodological reasoning alone. 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.

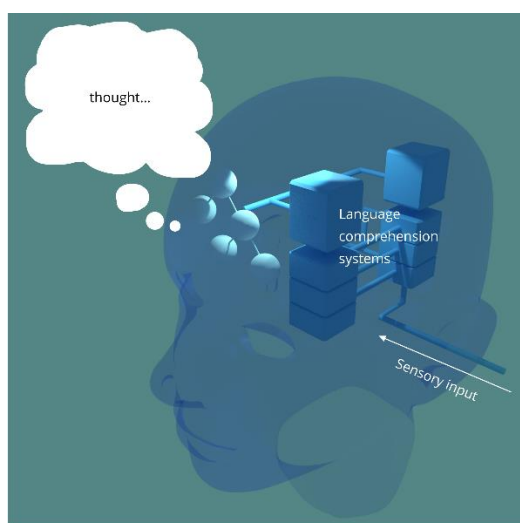
This document consists of two parts. The first, Section 2, provides the empirical framework. These are assumptions that are in my view justified both by empirical and practical concerns; they constitute the hypothesis put forward in this document. The second part, Section 3, goes into details concerning full formalization and the computational implementation of that particular 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. The status of the claims presented in Section 2 and Section 3 therefore differ in terms of their epistemological relevance: Section 2 describes a substantial empirical hypothesis, whereas Section 3 is concerned with details leading to full formalization and simulation, many of which are inessential to the empirical hypothesis.

The term “linear phase parser” comes from the fact that the parser processes the incoming sentence incrementally, from left to right, on a phase-by-phase basis.

## 2 The framework

### 2.1 Introduction

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 to accomplish such task without external information, such as context or other help, 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 *parser*, and the whole process that contains the parser is called *language comprehension*. Both are ultimately embodied as neuronal systems of the human brain that interact with each other and with other cognitive systems. The overall idea is sketched in Figure 9.



*Figure 9. Language comprehension includes a parser that implements a mapping from sensory input (linguistic expressions received through sensory systems) into sets of meanings (thoughts, ideas, etc). The parser is implemented by networks (loosely “modules”) in the human brain, some of which are dedicated to the processing of language, others which are used for other purposes in addition.*

Some sensory inputs map into meanings that are hard or impossible to construct, 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 or graded classification in terms of grammatical, marginal and ungrammatical sentences. Grammaticality and marginality 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 has no reason to focus on ungrammatical sentences in the first place, as there is no reason to “parse”

ungrammatical sentences as such. A realistic, observationally adequate parser must nevertheless capture them correctly.

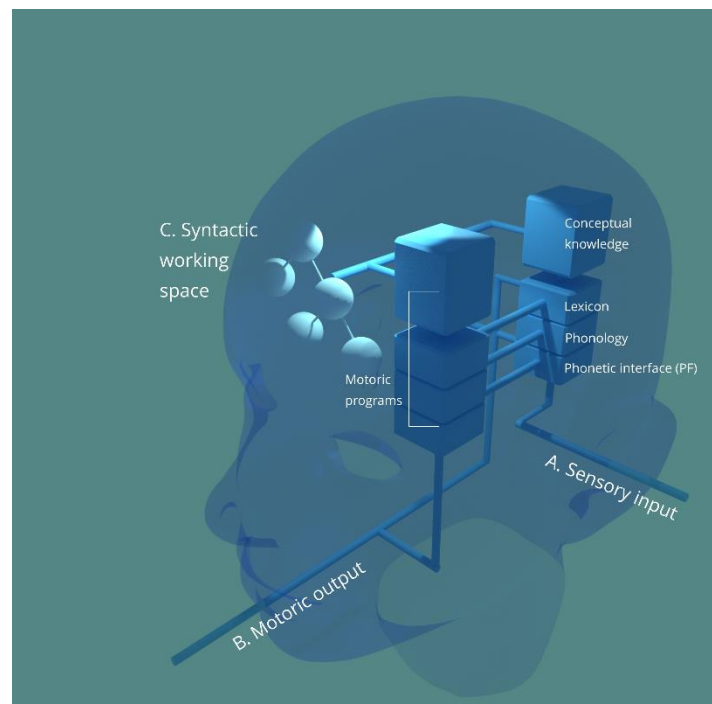
Some aspects of the parser are language-specific, others are universal and depend on the linguistic architecture and thus on the biological structure of the brain. A universal property can be elicited from the speakers of any language. For example, it is a universal property that an interrogative clause 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 required for the acquired aspects constitute the fixed portion of the parser, whereas particular 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*. Stated in this way, the notion of explanatory adequacy is relativized against a theory of language acquisition, itself an empirical issue; but there is no way for settling these questions in any other way.

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 here interested in a theory of language comprehension, perception and parsing that is explanatory.

The distinction between linguistic *competence* and linguistic *performance* is usually made so that the former refers only to the properties of the characteristic function from inputs into grammaticality judgments, hence “knowledge of the rules of language,” whereas the latter involves also the actual processes that implement the said characteristic function (see Figure 9). The emphasis is on the phrase “involves also”: the parser, or a broader theory of language comprehension, must embody a theory of competence (irrespective of whether that theory is in any way general, plausible or useful). It would be a mistake to infer from this truism, however, 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”, whatever that means) the motoric circuits and vice versa, it is clear that language perception does not force the listener to carry out the corresponding motoric programs in an overt way; 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 brain with systems that are ultimately responsible for the perception of the same units. The hypothesis is illustrated, in a nontechnical and sketchy way, in Figure 10.



*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 are connected with the same syntactic working space utilizing core computational information processing operations (Merge, Move, Agree) delineated later in this paper. 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 much simplified, seems to me to be at least consistent with what we know from recent 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 or help. 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, musical or ambient noise), it will be interpreted in terms of phonemes, syllables, surface morphemes and features, phonological words, lexical items (the posterior systems in Figure 10), 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 an interpretation to the sensory input that has no basis in the input. This would correspond to an interpretation that is assigned either randomly or to every input. Both are irrelevant. It does not follow from this, of course, that there are no non-stimulus-driven top-down processes, only that the more abstract interpretation must rely, at least in part, on those made at an earlier stage. In fact, what evidence we have currently suggest that while top-down influences exist, they are not necessary and can, therefore, be ignored if they do not constitute a specific topic of the inquiry. It does not follow, of course, that we must ignore them.

Consider a simple example, that of recognizing the word *mouse* from an input stream. Suppose we hear somebody uttering the sentence *the cat was chasing the m...* such that the last word was masked by an occurrence of a sudden noise. Most speakers would be able to guess that the last word is *mouse*. This would be an example of a top-down process: incomplete background information is used as a cue to form a guess, the latter which might turn out to be right or wrong (wrong if the sentence was *the cat was chasing the mice*). The operation is said to have a cognitive top-down orientation, because here we use abstract cognitive expectations to supply the information that is missing from the sensory object. These top-down expectations are perhaps used even if nothing were missing from the sensory object in the first place. But notice that such examples work only if we assume that the preceding words were recognized correctly. This means that the required information *was* available in the sensory input: we access the relevant encyclopedic information about cats and mice on the basis of the words and their mutual relations in this sentence.



Certain properties of this language perception pipeline are strictly necessary. The system must be able to recognize and decompose words and, when a word is not recognized, assign it 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, 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 before they may be interpreted semantically. 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 is 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 |                                |

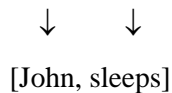
Similarly, we know from neurolinguistic and psycholinguistic studies that speakers react immediately to violations of this type. This requires that the speaker is able to create a syntactic representation or interpretation for the input sentence and, by examining the properties of the resulting structure (or noticing its absence), is able to judge that the clause is ungrammatical and violates the syntactic rules of the language. 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, that is able to *combine* the incoming words (or lexical items) into larger units. Let us next examine what kind of properties Merge must have in order to operate based on the sensory input alone.

## 2.2 Merge

Linguistic input is received by the hearer in the form of sensory stimulus. That input can be thought of as a one-dimensional string  $\alpha, \beta, \dots$  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. For example, if the input consists of two words  $\alpha + \beta$ , Merge will yield  $[\alpha, \beta](2)$ .

(2) John + sleeps.



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. 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.  $[[\text{John } \textit{furiously}] \text{ sleeps}]$     b.  $[[\text{John}, \text{sleeps}] \textit{furiously}]$     c.  $[\text{John } [\text{sleeps } \textit{furiously}]]$

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.

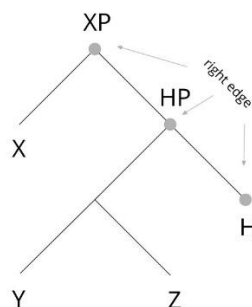


Figure 2. The right edge of a phrase structure. These three nodes are all possible attachment sites for an incoming word. X, Y and Z are not. This condition follows from the assumption that linearization in language production is always top-down/left-right.

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.

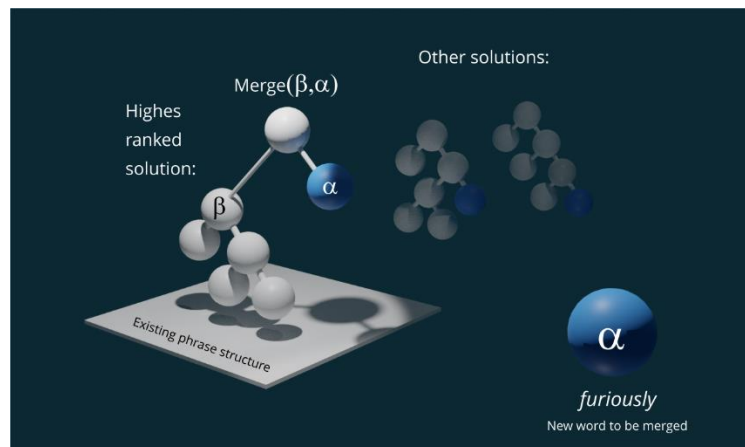


Figure 2. All possible merge sites of  $\alpha$  on the right edge of the existing phrase structure  $\beta$  are ordered and then explored in that order. This operation takes place in the syntactic working space, see Figure 10.

Both solutions (a) and (c) are “countercyclic”: they extend the phrase structure at its right edge, not at the highest node. A countercyclic operation 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 3.3.1). This type of derivation is often 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.”

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).

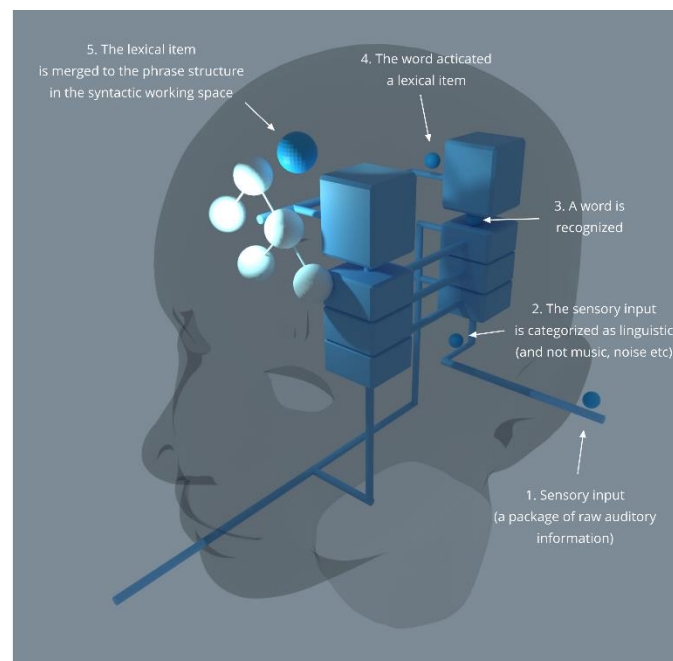
(6) [John [[sleeps furiously] γ]

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. For present purposes, it is extremely 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*.

As this discussion shows, the language comprehension perspective requires that we make several nontrivial and nonstandard assumptions concerning Merge. I do not know any way to avoid these (or some very similar) assumptions if we assume that language comprehension is incremental and that it uses the core computational operations of the human language.

There are at least two ways to think about what these changes mean. 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 on the basis of 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 eventually “converge” into some core set of assumptions or 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.



*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.*

### 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, 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 fact that *admire* has the lexical feature [*!COMP:D*] can be then used by the merge right operation to create a ranking: when *Mary* is consumed, the operation checks if any given merge site allows 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        Node accepted and prioritized.

Feature [*COMP:L*] means that the lexical item *allows* for 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.

There is a certain ambiguity in how these features are used. When the parser-grammar is parsing an input, it will use these features to rank the solutions. They cannot filter out solutions, because the constituency relations could 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. 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 semantic or conceptual-intentional systems.

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 (*Mary* has a lexical selection feature [ $-\text{COMP:ADV}$ ]). We are left with the two options. Let us examine the two options (9).

(9)

- a. [[<sub>S</sub> John [admires Mary]] furiously]
- b. [John [<sub>VP</sub> 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 by providing the adverbial with a lexical selection feature which determines what type of specifiers it is allowed or is required to have. I will call such features *specifier selection features*. A feature [ $\text{SPEC:S}$ ] favors solution (a), whereas [ $\text{SPEC:V}$ ] favors solution (b). If the adverbial has both features or has neither, then the 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 proper specifier-head relations in the output. Therefore, we use specifier selection to guide the parser in selecting left sisters, and then later use them to verify that the output contains proper specifier-head relations. To illustrate, consider the derivation in (10).

- (10) John's brother      admires
- ↓                      ↓
- [John's brother]    T(v, V)=finite tensed transitive verb

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 have this:

- (11) John's    brother    admires    Mary
- ↓                      ↓                      ↓
- [[John's brother] [T(v,V) DP]]

The grammatical subject occurs in the canonical specifier position of T(v,V). It is extremely important to keep in mind that in this framework most configurations established during 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 for specifically 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).

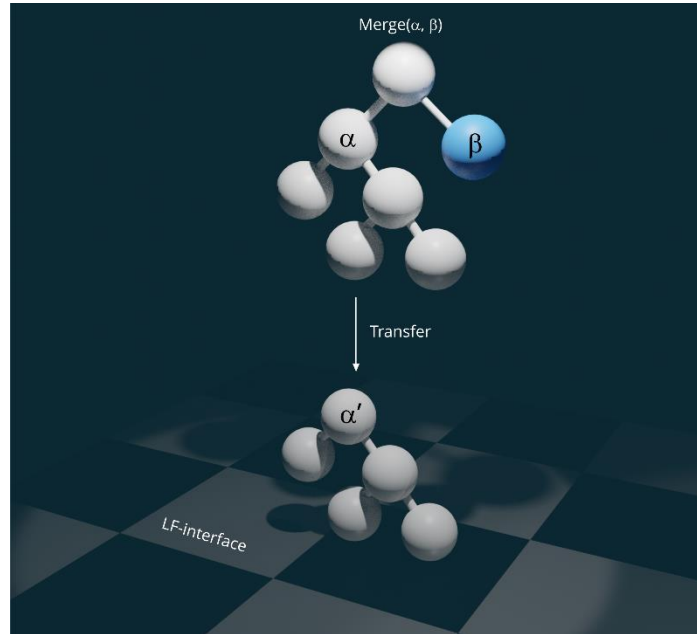
- (14) John's    +    mother    +    admires    +    Mary  
       [John's        mother    ]    ←Throw away!  
       [                        ]    +    admires    +    Mary    (Already packaged: “John's mother”)

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 further. Any merge site that leads into an ungrammatical or uninterpretable left branch can be filtered out as either unusable or 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 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 ‘prepare’ 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 drops a candidate left branch phrase structure  $\alpha$  to the LF-interface. The transfer operation implements several computational operations, and the resulting phrase structure  $\alpha'$  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.*

The nature of the LF-interface is not a trivial matter. It is typically conceptualized as the interface between syntax and semantics. We this really means 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 same way as a native speaker interprets the sentence. This is the technical interpretation used in the present work. I do not view it as a “language thought” that has to be utilized in all acts of conceptual thinking, or as a universal conceptual starting point for the generation of linguistic expressions. If transfer functions like a reflex, then it is possible to generate thoughts by creating the linear left-to-right string associated with the Merge Right operation; there is no need to generate them from the LF-interface. The principles regulating LF-objects are still formal in nature.

## 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 any complex constituent, such as *John's mother*. A recursive labeling algorithm (16) is postulated for this purpose. The intuitive function of the algorithm is to “search for the closest possible primitive head from the phrase,” which will then constitute the label. Here “closest” means from 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 XP is a complex phrase. Then

- a. if the left constituent of XP is primitive, it will be the label; otherwise,
- b. if the right constituent of XP is primitive, it will be the label; otherwise,
- c. if the right constituent is not an adjunct, apply (16) 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 will still be primitive, according to these definitions (this becomes relevant later: it is like a head with a clitic on it). 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.”<sup>2</sup> 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: its structure is [D V]. The primitive left constituent D will be its label. There are at least three ways to solve this problem.

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

---

<sup>2</sup> 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. However, it leads into empirically correct results in some cases and is therefore preserved.

as exceptions. They are not exceptions, however: there is nothing anomalous about (17). The representation 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)     John +    sleeps.  
             ↓            ↓  
             D    N    sleeps  
             ↓    ↓    ↓  
             [VP[DP D   N] sleeps]

It is important to notice 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 verb phrase. The outcome will be the same if the verb is transitive. The structure and label are provided in (19).

(19) [VP[DP D N] [ V<sup>0</sup> [DP D N]]

## 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 Merjaa *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 the basic labeling. The problem is to define what this type of ‘free attachment’ means.

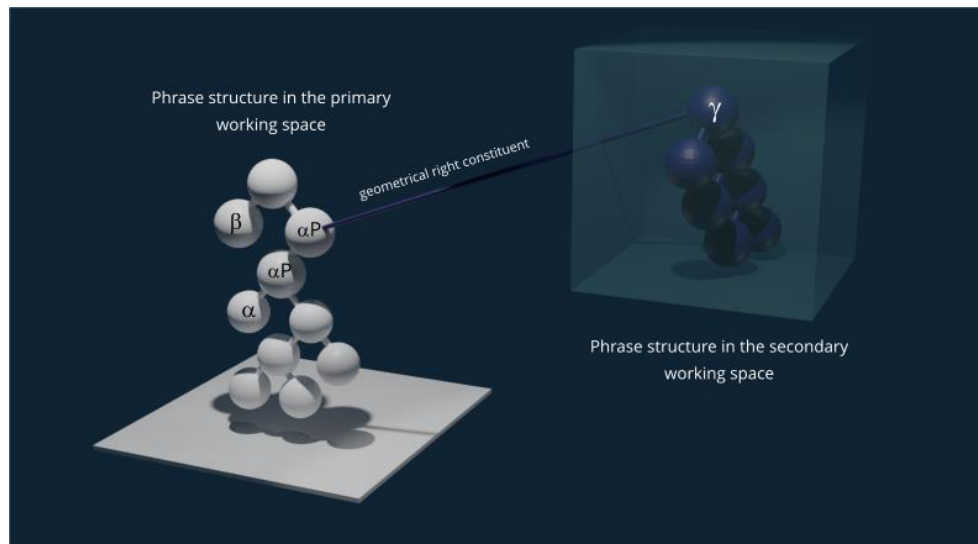
It is assumed here that adjuncts are geometrical constituents of the phrase structure, but they are stored in a “secondary syntactic working memory” and are invisible for sisterhood, labeling and selection in the main working memory. Thus, the labeling algorithm specified in Section 2.5 ignores adjuncts. 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)  $H^0$  [<sub>VP</sub> John [<sub>VP</sub> [<sub>VP</sub> admires Mary] <<sub>AdvP</sub> 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 <<sub>AdvP</sub> 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 [<sub>VP</sub> [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.



*Figure 3. Adjuncts  $\gamma$  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  $\beta$  therefore sees  $\alpha P$  as its sister.*

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. On the other hand, these assumptions also entail that they 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 a 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.

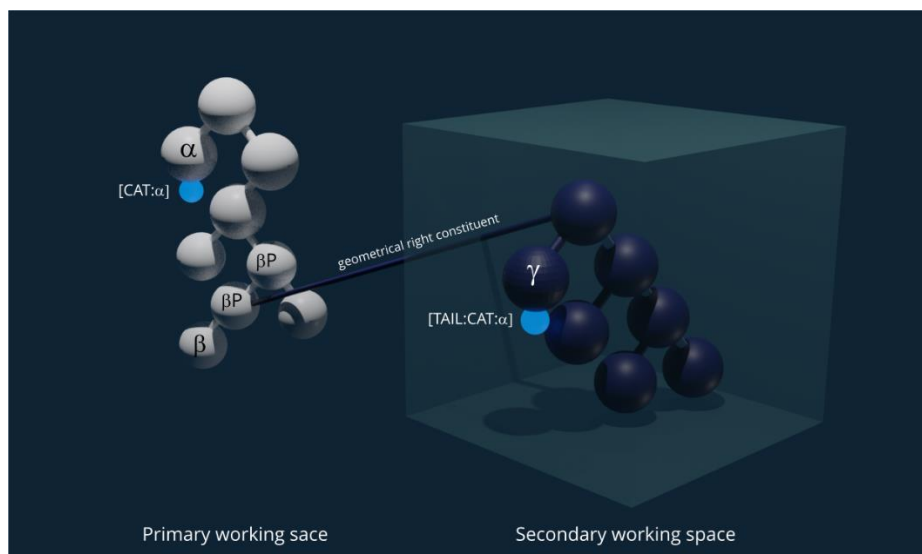
*(23) Condition on tail-head dependencies*

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 their right-adjoined or extraposed positions in the 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  $\gamma$  of the adjunct with the tail-head feature  $[TAIL:CAT:\alpha]$  must be c-commanded by a head that has feature  $[CAT:\alpha]$ . In the more traditional bottom up theories,  $[TAIL:CAT:\alpha]$  is the goal feature requiring checking, whereas  $[CAT:\alpha]$  is the probe feature. Another legitimate condition is if the probe feature were at  $\beta$ , because the adjunct would then be inside  $\beta P$ .*

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 2.8.6.

The fact that constituents can satisfy some condition in a position different from the one inferred from the surface string means that most grammatical rules checking features must be specified in relation to a *stage* in the derivation. An important property of the rule (23) and many similar rules is that each constituent must satisfy such rules only once, or in one position. If  $C_1, \dots, C_n$  is a chain of positions of a constituent in a chain formed by grammatical operation(s) such as movement, then the weakest possible criterion is that *one* member of the chain must satisfy the condition, but it does not matter which one. A stronger condition, the one that is relevant for (23), is that the last (LF-interface) position must satisfy the condition. The intuitive motivation is that the condition is relevant at LF for the semantic interpretation, which must pair adverbials with events expressed by verbs in order to arrive that proper semantic interpretation. In the parser-grammar framework, the representation that is generated last is the one that is also fed to LF. Thus, when the condition is checked at LF before the phrase structure is passed to systems responsible for extrasyntactic interpretation, the copies or traces (if any) are ignored and only the last position will be checked. This is not always the case, however. The EPP property (Section 2.7) of some head need not be checked by the last element in a chain: it can be the first, second, last, or any other member in between.

## 2.7 EPP

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 2.8.3). This constitutes a sufficient (but not necessary) feature for reconstruction; see 2.8.3. Language uses phrasal movement, and hence an unselective specifier feature, to represent a null head (such as C) at the PF-interface (Section **Error! Reference source not found.**). Corresponding to [SPEC:\*] we also have [!COMP:\*], which is the property all functional heads have by definition.

## 2.8 Move

### 2.8.1 Movement and error tolerance

The problem of displacement gets a slightly different interpretation under the comprehension perspective. The problem is sometimes attributed specifically to the generative framework, but it is a general empirical linguistic problem. From the point of view of language comprehension, the problem is that we cannot rely exclusively on linear order in interpreting the syntactic and semantic relations between the elements of the clause. Consider an input sentence such as (24) and some possible first pass parses of the input provided in (25) (in pseudo-English for simplicity) that uses nothing but linear order and selection.

- (24) Ketä Pekka sanoi että Merja ihailee \_\_? (Finnish)  
who.par Pekka said that Merja admires  
'Who did Pekka say that Merja admires?'

(25)

- a. [who][[Pekka][said that Merja admires]]
- b. [who Pekka][said that Merja admires]]

To interpret (24) correctly it is mandatory that the parser associates the initial interrogative pronoun with (i) interrogative force and (ii) the thematic role of the patient of the embedded verb *admire*. The interrogative pronoun serves two roles: it marks the scope of the question and represents the patient of the embedded verb.

Scope marking (i) has functional motivation: because the interrogative head (or the interrogative feature marking the clause as an interrogative) is phonologically null in Finnish, the interrogative pronoun signals its presence at the PF-interface. Therefore, it will be assumed in this work that the dislocated interrogative pronoun is used to *generate* an interrogative feature to the clause that may then be selected by a higher verb

(e.g. *wonder, ask*); this is the inverse of the feature checking operation of the bottom-up theories. If a separate question particle existed with phonological content, the *wh*-element would have the option of remaining in situ. In general, movement allows the language faculty to express and generate features that are not visible at the PF-interface. Notice that the ‘criterial feature’ (*wh*-feature in this case) that is generated is present in the movement element, so the moved element can be seen as ‘carrying’ it to the front.

Let us consider (ii) next. The interrogative pronoun must be associated with the patient role of the embedded verb *admire*. Syntactically, the association is motivated by the fact that if left undone, the lexical selection feature of the verb will not be satisfied: it appears as if there were nobody that Mary admires (thus, the clause would end *Mary admires*). Semantically, the interrogative sentence is asking the identity of the object of admiration. There must therefore be a way to represent the fact that the patient of admiration is the unknown element (i.e., a variable). The moved interrogative pronoun serves also these roles. It has a *wh*-feature in it that is interpreted as representing the unknown constituent, and which must be linked (bound in some way) with the interrogative feature or the scope marker at the beginning of the clause to represent the scope of the question. From the point of view of language comprehension, the fact that the pronoun represents the variable is directly available at the PF-interface and can be read off from the overt form of the pronoun (thus, it is an interrogative pronoun), but the fact that it represents the patient of the verb *admire* is not; linear order would compel the system to associate it with elements in the main clause.

From these facts alone it follows that the first pass parse (26) must be subjected to some computational operation(s) that accomplish (i-ii) above.

(26) [who][[Pekka][said that Merja admires]]

Those operations must associate the fronted interrogative pronoun with an interrogative force and the scope of the question, and they must link the interrogative pronoun with the patient role of the verb *admire* (and not *say*, for instance). The whole sequence of the required operations will be called *transfer* in this work, because it will link the surface structure generated by first pass parse with a “deep structure” representation that represents (i-ii) explicitly. In the present work, the “deep structure” will always be a LF-object, a representation that will be sent off to the conceptual-intentional system for full semantic interpretation.

In the present work it is assumed, following the generative theory, that the interrogative pronoun is literally reconstructed or copied to the position in the structure in which it will satisfy the (intended) lexical selection features. This assumption means that we do not need to process an anomalous sentence fragment *Merja admires* that lacks a mandatory object; instead, after reconstruction the argument will be available at its canonical position at which it is selected, case marked, and at which it is interpreted semantically as the patient of the verb. The result will be (27).

(27) Who C(wh) Pekka said that Merja admires (who)



This captures the intuition that the interrogative pronoun serves two roles in the clause: it marks the scope of the interrogative and constitutes the patient of the embedded verb. It is literally at two positions, only one of which is pronounced. Both the generation of the interrogative force/scope (i) and the reconstruction (ii) are part of transfer that maps the first pass parse (s-structure) into an LF-object. The relationships between the three representations discussed so far are shown in (28). The term “s-structure” refers here to the phrase structure representation resulting from the first pass parse.

(28) Linear order  $\leftarrow$ linearization, first pass parse $\rightarrow$  s-structure  $\leftarrow$ transfer $\rightarrow$  LF-interface object

Notice that the assumption that the interrogative pronoun is copied into the canonical position is an empirical assumption. The same result could be achieved by some other mechanism that links the filler with the gap. Looked from this perspective, the only thing that matters is that a formal, explicit filler-gap dependency is posited; if that condition is not satisfied, there is nothing more to say.

We can now consider certain additional constraints that any comprehension theory must satisfy. One concerns the causal input configuration that triggers some operation during transfer. The operation cannot reconstruct or generate features willy-nilly. For example, it cannot reconstruct *Pekka* into the complement position of *admire*; it must target *who*. The second concerns the output configuration that defines what the operation will do. It should not reconstruct *who* into the complement position of *say*, it must go to the complement position of *admire*. The following general input condition turns out to be virtually universal when the issue is examined from the point of view of language comprehension:

(29) *Input condition (nontechnical definition)*

Trigger an operation during transfer if and only if the first pass parse has an element  $\alpha$  in a position  $\beta$  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. These problems could result from an error in the production or from a mistake made by the listener; in the example (24) the mistakes were intentional. These were two such ‘errors’ in the interrogative clause (24): (1) the finite verb was associated with two DP specifiers *who* and *Pekka*, one of which is not selected or licensed by anything, and (2) the embedded verb *admire* lacks a complement and cannot be interpreted without knowing who is it that is being admires. Both errors trigger transfer according to (29): the first error leads into the generation of the supporting interrogative head C(wh), and the latter to an attempt at copying the interrogative pronoun to the complement position. This gives us the output condition:

(30) *Output condition (nontechnical definition)*

Transfer will attempt to generate a legitimate LF-object.

I will hypothesize that the function of transfer (and hence also movement reconstruction) is to render language comprehension insensitive to errors, and that the production of an interrogative clauses and other similar constructions in which transfer becomes a systematic feature of language are *consequences* of the existence of a more broad error tolerance mechanism. In other words, speakers are generating errors intentionally in order to open up options (null heads, dual roles) not otherwise present.

### 2.8.2 *General properties of Transfer*

All or most operations performed during transfer follow the same general logic that will be discussed in this section before looking at each operation specifically. An error is recognized by (i) detecting a *selection violation* and/or by noticing that (ii) 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 a ‘dropping’ operation, in which the illegitimate object  $\alpha$  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 (in terms of i-ii) is selected. The operation is simply an error correction mechanism: it tries to find a proper position for the orphan element. Dropping cannot visit left branches, which are phases, 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.

There are a few steps that do not fit into the above description, not at least in any obvious sense. 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 from 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. My view is that the system is viewing the criterial feature as an ‘extra’ component whose presence is not motivated; if so, then also the occurrence of a non-selected or unmotivated feature could trigger error recovery. Let us assume that this constitutes a third criterion.

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  $[_T \emptyset [_V \emptyset V]]$ . Because  $[\emptyset X]$  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  $[\emptyset X]$  is illegitimate or uninterpretable structure at LF. If so, we add a fourth condition: illegitimate constituency structure.

All in all, there are four illegitimate conditions that trigger error recovery:

- (31) Some input conditions
- A. Violation of specifier or complement selection
  - B. Violation of probe-goal
  - C. Presence of an unselected or unmotivated feature
  - D. Violation of constituency

With these general points being made, we can now examine individual operations.

### 2.8.3 *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 (32) 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.

- (32) John        admires    Mary
- ↓                ↓                ↓
- [John        [admires Mary]]

Example (33) shows a variation in which this is no longer the case.

- (33) Ketä        Pekka        ihailee    \_\_\_? (Finnish)
- who.par    Pekka        admires
- ‘Who does Pekka admire?’

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

- (34) [who][Pekka admires]

The clause violates selection twice: first due to the occurrence of the two specifiers at the left edge and then due to the missing object of *admire*. The double specifier problem is fixed by generating a head between them (35).

- (35) Ketä        C(wh)    Pekka        ihailee    \_\_\_?
- wh            C(wh)
- who.par                Pekka        admires
- ‘Who does Pekka admire?’

The mechanism has two computational steps that must be distinguished. The first is the recognition that a head is missing. That is inferred (again, in this particular example) from the existence of the two specifiers of the T/fin head. 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 and thus recover from the ‘error’ in which the head remain unpronounced.

The interrogative phrase must be reconstructed back to its canonical LF-position to satisfy the complement selection for the verb *admire*.

(36) Who        does        John        admire    \_\_\_?  
                  ———Reconstruction—————→

Reconstruction works by copying the element that appears in a “wrong” position and then by dropping it into the same structure, in this case dropping begins from the sister of the 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 thus be targeted by another operation later during transfer. If the position is found, it will be copied there.

Transfer 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( $\alpha$ ,  $\beta$ ) and leads to the transfer of  $\alpha$ . These two conditions are expressed by (37)(Brattico and Chesi 2019).

(37) Merge( $\alpha$ ,  $\beta$ ) or  $\alpha$  is completed  $\leftrightarrow$  Transfer( $\alpha$ )

One question that still remains without explicit answer is the explicit trigger for the operation that reconstructs the interrogative pronoun to the complement position. Notice that after the 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 begins searching for a suitable element. In the present implementation, it is assumed that error recovery is always 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 position will be found during dropping. The existence of the EPP feature has always presented a puzzle. According to the hypothesis proposed here, it consisted an unselected and uninterpretable (hence wrong) specifier position that has been canonized in a language.

#### 2.8.4 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.

#### 2.8.5 Head reconstruction

Many heads occur in noncanonical positions in the input string. Consider (38).

- (38) 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<sub>inf</sub> (-a-) and V (*nukku-*). Morphology extracts this information from the phonological word and feeds it to syntax (39). Symbol “#” indicates that there is no word boundary between the morphemes/features.

- (39) C(-kO) + #T<sub>inf</sub> + #V +    Pekka + ajatteli + että +    hänen + pitää  
       C            T            V        Pekka    thought    that        he            must

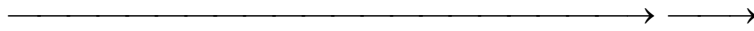
The individual heads are collected together in syntax into one complex head. The incoming morphemes are “stored” into the right constituent of the first morpheme, a process that resembles cliticization. Thus, if syntax receives a word-internal morpheme, it will be merged to the right edge of the previous morpheme: [<sub>α</sub> ∅ β]. Notice that by defining “complex constituent” as one that has both the left and right constituent, [<sub>α</sub> ∅ β] comes out as “primitive constituent” and can therefore constituent the head of a projection. The linear sequence C-T/inf-V becomes [<sub>C</sub> ∅ [<sub>T/inf</sub> ∅ V]], and this is what gets originally merged (40).

- (40) [<sub>C</sub> ∅ [<sub>T/inf</sub> ∅ V]]    Pekka    ajatteli    että hänen    pitää    \_\_\_\_  
                                  Pekka    thought    that he        must

Representation (40) is not a legit LF-object. The first constituent [<sub>C</sub> ∅ [<sub>T/fin</sub> ∅ V]] cannot be interpreted, and the embedded modal verb *pitää* ‘must’ lacks a complement. Both problems are solved by head

reconstruction which drops T from within C into the structure. This is done by finding the closest position in which T/fin 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<sub>inf</sub>.

(41) C{T<sub>inf</sub>{V}} Pekka ajatteli että hänen [pitää [T<sub>inf</sub>{V} V]].  
Pekka thought that he.gen must



The operation fixes problems with cluttered heads in the input. Accordingly, the reason the heads are cluttered together in the first place is because language takes advantage of the error recover mechanism and is thus able to ‘compress’ several words into one. It is conceivable that this achieves some type of optimization at the PF-interface. The process is otherwise similar to phrasal movement reconstruction: it considers the cluttered components of the complex head to be in a wrong position (which they are, see (38)) and attempts to drop them into first positions available in which they could create a legitimate LF-object.

#### 2.8.6 Adjunct reconstruction

Consider the pair of expressions in (42) and their canonical derivations.

(42)

- 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  
↓                ↓                ↓                ↓                ↓  
[Merja [ asked [we [to.admire Pekka]]]  
‘Pekka asked us to admire Merja.’

Derivation (a) is correct, (b) is incorrect. This is because the thematic roles are identical 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. The system introduced so far has 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 (43).

- [nom] is checked by +FIN,
- [acc] is checked by v (ASP),
- [gen] is checked by -FIN
- [par] is checked by -VAL.

(44) [ $\langle \text{Merjaa} \rangle_2$  T/fin [  $\text{---}_1$  [käski [meidän [ihailla [  $\text{---}_2$   $\langle \text{Pekka} \rangle_1$ ]]]]]  
+FIN  $\longleftrightarrow$  [nom] -VAL  $\longleftrightarrow$  [par]  
Merja.par asked we.gen to.admire Pekka  
‘Pekka asked us to admire Merja.’

## 2.9 Ordering of operations

(45) Merge  $[\alpha, \beta] \rightarrow$  Transfer  $\alpha$ : reconstruct heads  $\rightarrow$  reconstruct adjuncts  $\rightarrow$  reconstruct A/A'-movement

The whole sequence is performed in a one fell sweep, mechanically like a reflex. It must be “costless” for the brain; otherwise languages would gravitate towards super-analytical systems that avoid them. The reason it does not “cost” anything is because such error recovery must be part of any language production and comprehension system that has to work under less than optimal circumstances.

## 2.10 Lexicon and morphology

Most phonological words enter the system as 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 (46). Notice the reversed order.

(46) Nukku-u-ko + Pekka → Q + T/fin + V + Pekka  
 sleep-T/fin-Q Pekka ↓ ↓ ↓ ↓  
 sleep#T/fin#Q [Q [T/fin [V Pekka]]]

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.<sup>3</sup> This is illustrated in Figure 6.

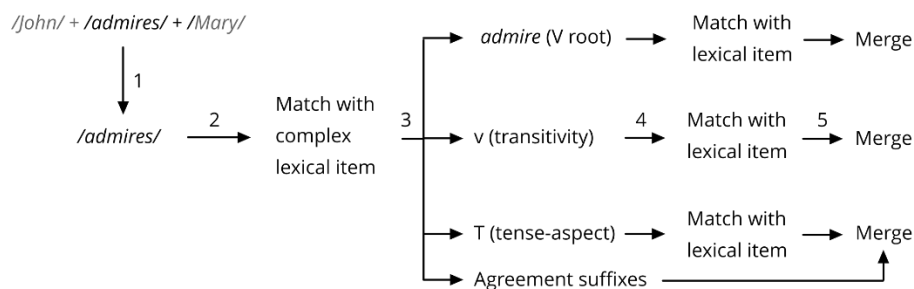


Figure 6. A schematic overview of the structure of the lexicon and the computational operations involved in lexical decomposition. The whole pipeline contains the following steps: (1) consume a phonological word from the input; (2) match the word with a lexical item, or a (ranked) list of lexical items if it is ambiguous (in

<sup>3</sup> 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).



*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. An inflectional feature F in a sequence  $m_1\#...m_i\#F$  will become a feature of the preceding morpheme  $m_i$ : *Merja-a* ‘Merja.par’ = N#D#par = [D(par, def...) N].

Lexical features emerge from three distinct sources. One source is the language-specific lexicon, which stores information that is specific to a particular lexical item in a particular language. For example, the Finnish sentential negation behaves like an auxiliary, agrees in  $\phi$ -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 as ‘feature implications’. These redundancy rules make up a sort of ‘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.<sup>4</sup> Lexical features constitute an unstructured set, but the features themselves can have a ‘type:value’ structure. For example, labels are provided as values (N, V, A, ...) of the type CAT(egory), so the actual features are of the form (CAT:L).

---

<sup>4</sup> A third source of lexical features comes from parametric rules. For example, in Finnish most heads that have the [SPEC:\*) feature also exhibit  $\phi$ -agreement in some context (e.g., prepositions, noun heads), whereas the same functional heads in English have neither (Brattico et al. 2019). Such generalizations could be stored in the language-specific lexicon, but this would be redundant to the extent that a general rule is at stake. Because the rule is still language-specific, it cannot be part of the list of universal redundancy rules. It is unclear what the source and implementation of these rules is; they look like language-specific parts (or parametrizations) of the mini-grammar defined by the redundancy rules, but it is unclear what the term ‘language-specific’ really means and what it means to have ‘parameters’. These regularities are, therefore, stipulated in a third component of the lexicon.

## 2.11 Argument structure

The term “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, 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 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 contain a [!SPEC:P] feature. Ideally, verbal lexical entries should contain a label for a verb class, and that feature should be associated with its feature structure by 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 (see Figure 4). 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 (47). 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.

(47)

- |    |        |        |           |
|----|--------|--------|-----------|
| 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.

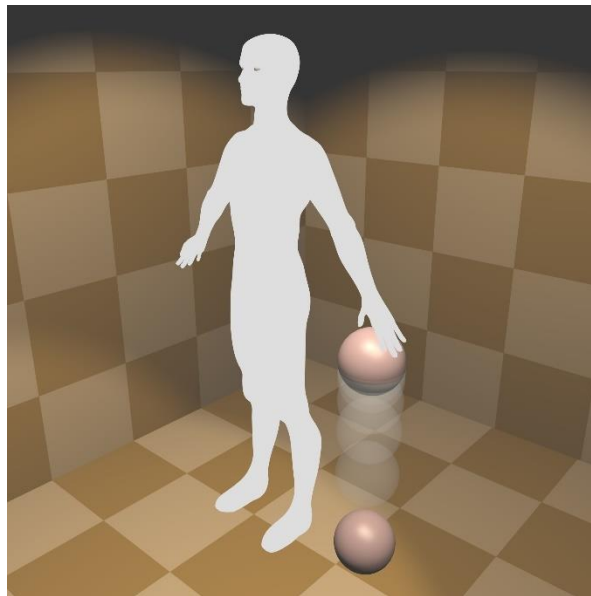


Figure 8. A non-discursive, perceptual or imagined representation of the meaning of the sentence *John dropped the ball*.

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 some 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 or the conscious representation of it. 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. Therefore, formal selection is an autonomous process. It is possible that these features exist for the purposes of making language production and comprehension possible, providing 'instructions' for the parsing derivation.

## 2.12 Agreement, null subjects and control

Inflectional  $\phi$ -features of predicates, such as verbs, are extracted from the input and embedded as morphosyntactic features to the corresponding lexical items (48).

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

These features are then used to saturate or value the unvalued  $\phi$ \_ features of lexical items. For example, when deriving a sentence such as (49), the lexical constituent *admire* will contain two features relevant for agreement: an unvalued  $\phi$ -set designated by  $\phi$ \_ (from the lexicon), and the agreement suffixes {3sg} extracted from the input.

(49) John      admire+s      Mary  
      [John    [admire    Mary]]  
               $\phi$ \_ = 3sg

More generally, during Transfer, an *agreement reconstruction* takes place after all movement has been reconstructed that tries to value  $\phi$ \_ for each head possessing the feature. Agreement reconstruction first tries to value  $\phi$ \_ 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  $\phi$ -set obtained from the input (if any). The  $\phi$ -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  $\phi$ -suffixes are unambiguous and are treated as ‘pronominal’ enough to value  $\phi$ \_, which leads into the familiar pro-drop profile: the subject can remain covert if an unambiguous  $\phi$ -set is present at the verb. If nothing can value  $\phi$ \_, 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 (50), the verb to leave cannot value its  $\phi$ \_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.”

(50) John wants to leave $_{\phi\_ = \text{John}}$ .

If no antecedent is found, a generic or arbitrary interpretation results, in which the argument of a predicate refers to ‘people in general’ (51).

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

The unvalued  $\varphi_{-}$  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  $\varphi_{-}$  – which is saturated by means of valuation. Agreement and control, then, constitute a reflex of saturation and are both a ‘repair’ strategy.

### 3 Formalization and implementation

#### 3.1 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.

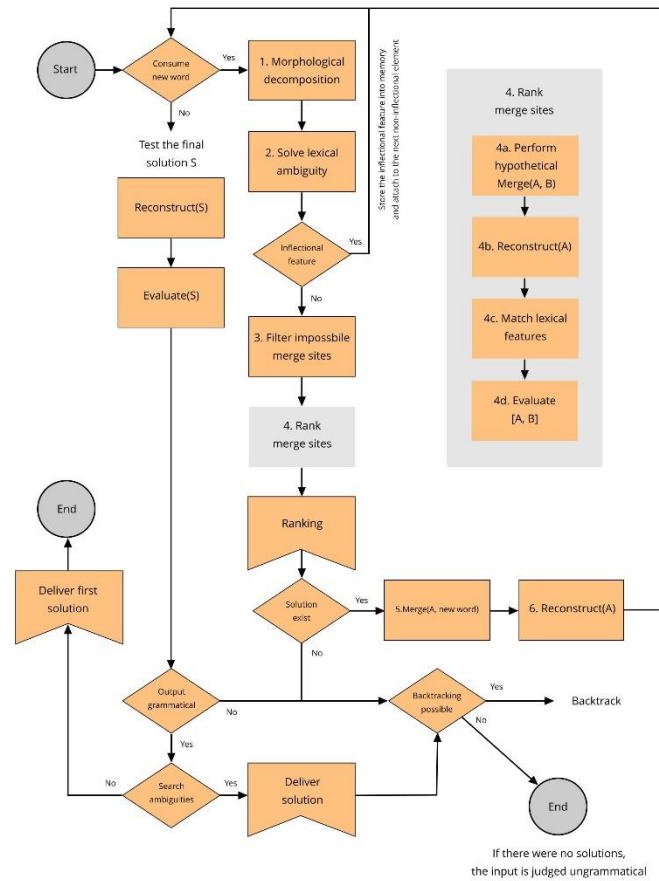


Figure 1. The structure of the generative linear phase parser-grammar language comprehension algorithm.

The individual components are commented in the subsections below.

## 3.2 The parser

### 3.2.1 *Phillips cycle*

The recursive parsing function takes the currently constructed phrase structure  $\alpha$ , a 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,  $\alpha$  will be transferred to LF. Transfer recovers from errors, reverse-engineers movement and agreement and performs LF-legibility tests to check that the output is semantically interpretable. If it passes all phases, the result will be accepted: the system has “understood” the expression.

Suppose word  $w$  was consumed. To retrieve properties of  $w$ , the lexicon will be accessed with  $w$  as an argument. This operation corresponds to a stage in real 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  $[w_1, w_2, \dots, w_n]$  that will be explored in some order (added to the recursion). 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. 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 contains a morphological decomposition of the word, if any. 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.

Suppose morpheme  $m$  was consumed. The morpheme must now 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 3.2.2). Each site from the ranking is then explored. For each site there are two options: if the new morpheme  $\alpha$  was part of the same word as the previous morpheme  $\beta$ , it will be “sink” into the word to create a complex head  $[\beta\emptyset, \alpha]$ . If not,  $\alpha$  will be merged countercyclically to the existing phrase structure, and the parsing function is then called recursively. If a ranked list has been exhausted without a legitimate solution, the algorithm will backtrack to an earlier step. Backtracking is rare in real-life conditions and only occurs during artificially created garden-path scenarios.

### 3.2.2 *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. (1) If the new morpheme was inside the same word as the previous one, all other solutions expect merge to the complement will be filtered out. (2) Primitive sites that do not accept any type of complements ( $-\text{COMP}^*$ ) are filtered out. (3) If a site is not

primitive, it will be reconstructed silently (Section 3.3.2) and tested for LF-legibility. If the new morpheme is adjoinable (and could become right adjunct), then we do not know if  $\alpha P$  will be left branch in the final output and we abandon any attempts at rejection; otherwise, LF-legibility can be used. (4) In addition, failure to satisfy the probe-goal test, head integrity test or the criterial feature integrity test (Section 3.7) will lead into permanent rejection: these tests cannot be satisfied by any further operation. There could exist other criteria for filtering and rejection. On the other hand, these tests must be used conservatively, as any site rejected here will never be considered, and the whole parsing branch will be closed permanently.

*Ranking.* The ranking function weights each solution based on various criteria, with larger number indicating priority. 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 significantly, but it has no independent theoretical motivation; it can be regarded as a heuristic. The ordering does not affect solutions found by the algorithm, only efficiency and the first solution returned by the algorithm. Their justification is in their ability (or inability) to guide the parser efficiently to a solution. Suppose the new element is  $h$  and the site to be tested is  $S$ . The criteria used in the current version are the following:

(52) *Ranking criteria*

- a. Positive specifier selection (+):  $S$  matches for  $h$ 's specifier selection,
- b. Negative specifier selection (+):  $S$  matches for  $h$ 's negative specifier selection,
- c. Negative specifier selection of everything (−):  $h$  has  $[-SPEC: *]$ ,
- d. Infrequent specifier feature (−):  $S$  matches for  $h$ 's infrequent specifier selection (not used),
- e. Do not break existing head dependencies (−):  $[S\ h]$  would break existing  $\alpha^0\text{-}\beta^0$  dependency,
- f. Tail-head test (−): check if  $[S\ h]$  would fail  $h$ 's tail features,
- g. Complement test (+):  $S$  (or any morpheme inside  $S$ ) selects for  $h$  as complement,
- h. Negative complement test (−):  $S$  (or any morpheme inside  $S$ ) does not select for  $h$  as complement,
- j. Semantic mismatch (−):  $S$  and  $h$ 's semantic features mismatch,
- k. Left branch evaluation (−): Reconstructed  $S$  fails LF-legibility,
- l. Adverbial tail-head test (−):  $h$  has label Adv but fails tail-head test when c-commanded by T/fin,
- m. Adverbial tail-head test (+):  $h$  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. This heuristic rule works in practice and has no theoretical or intuitive motivation.

The ranking system adopted here has produced extremely efficient solutions for the empirical phenomena explored thus far. It is expected, however, that many more factors affect ranking, and that the above list is incomplete and partially wrong. These insufficiencies will emerge later when more empirical phenomena are explored.

### 3.3 Definitions for formal primitives

#### 3.3.1 *Merge: cyclic and countercyclic*

Simple Merge takes two constituents  $\alpha$  and  $\beta$  and joins them together to form  $[\alpha, \beta]$ ,  $\alpha$  being the left constituent,  $\beta$  the right constituent. Countercyclic Merge is a more complex operation. It targets a constituent  $\alpha$  inside an existing phrase structure and creates a new constituent  $\gamma$  by merging  $\beta$  either to the left or right of  $\alpha$ :  $\gamma = [\alpha, \beta]$  or  $[\beta, \alpha]$ . Constituent  $\gamma$  then replaces  $\alpha$  in the phrase structure, with the phrase structural relations updated accordingly. If  $\alpha$  is primitive and an adjunct,  $\gamma$  will be adjunct as well. Countercyclic Merge is not symmetric:  $\beta$  is attached to a node  $\alpha$  inside a complex phrase structure, not vice versa. An inverse operation of countercyclic Merge is *remove*, which removes a constituent from the phrase structure and repairs the empty hole left behind. 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. The guiding intuition behind this theory of Merge is that it constitutes an operation that manipulates phrase structures without being inherently limited in the nodes it may target, and thus represents “generalized transformation” in the more standard theory but is here interpreted more akin to an error recovery operation. There is, in my view, no way to avoid this generalization when we examine language comprehension.

Only Merge Right is used during the first pass parse when reading the input. Merge Left is activated during Transfer when wrongly positioned constituents are reconstructed and dropped.

#### 3.3.2 *Sister, specifier, complement*

Iterative application of merge creates a structural entity that is used to define basic grammatical relations. A *sister* of  $\alpha$  is the non- $\alpha$  daughter of the first branching node from  $\alpha$  that is not right-adjunct. In addition, right adjuncts don’t have sisters, so that they are “isolated.” It follows that in  $[\alpha, \beta]$  both  $\alpha$  and  $\beta$  are each other’s sisters. In  $[\langle\alpha\rangle, \beta]$  the same holds, but in  $[\alpha [\beta, \langle\gamma\rangle]]$ , the sister of  $\beta$  is  $\alpha$ . Right adjuncts are invisible for sisterhood. A separate relation of *geometrical sisterhood* takes them into account.

A specifier of  $\alpha$  is typically defined as a left phrase inside  $\alpha P$ . We define the notion of *local specifier* so that it constitutes the left phrasal sister of the node immediately dominating  $\alpha$ , with left adjunct phrases ignored. Furthermore, if no such local left phrase exists inside  $\alpha P$ , then  $\phi$ -features inside  $\alpha$  can constitute a ‘pronominal specifier’ if the  $\phi$ -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  $\alpha P$  plus pronominal elements inside  $\alpha$ . It is the same as the “edge.” It allows the parser to process pro-drop constructions and clitics. The specifier of a right head  $\beta$  is its complex left sister (i.e.  $XP$  in  $[XP, \beta]$ ). The relevance of local specifier is that it is the notion that enters into specifier selection. A more general definition of a *generalized specifier* is provided as well, which denotes any left phrase inside  $\alpha P$  together with any pronominal material inside  $\alpha$  if nothing else is found. Generalized specifier can be a left adjunct, but local specifier cannot. Thus,



the set of generalized specifiers of  $\alpha$  contains all left phrases inside  $\alpha P$ . From these definitions it follows that we could subsume both notions under generalized specifier and define local specifier as a special instance of the latter. This requires corresponding changes in every operation using either of these notions, so the reduction was left for future.

A *complement* of  $\alpha$  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.

### 3.3.3 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 phrases 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 they 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  $[\alpha, \beta]$  is  $\beta$ , but the right edge of  $[\alpha, \langle \beta \rangle]$  is  $\alpha$ . The intuitive motivation for this definition is that  $\beta$  is inside another syntactic working space. A *downstream* or *downward* relation follows this definition. The *left branch* of  $[\alpha, \beta]$  will always be  $\alpha$ . 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. Notice that adverbials are not  $A'$ -movement islands if they occur in the downward path, again the correct result empirically.

Consider sentence (53).

(53)	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.

### 3.3.4 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 exactly this way in the present implementation. For example, 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 (!COMP:T/FIN), but this runs into trouble in Finnish in which the sentential negation head can intervene between C and T(41).

- (54) C   Pekka   ei        nuku.  
       C   Pekka   Neg      T  
       ‘Pekka does not sleep.’

The problem cannot be solved by assuming Finnish-specific selection  $C \rightarrow \text{Neg}$  and  $\text{Neg} \rightarrow T$ , because the negation is not obligatory, and adding the rule  $C \rightarrow T$  would still allow the parser to create illegitimate C-H configurations, H being any head. We need a rule that forces the  $C \rightarrow T$  connection, but if we posit it as obligatory complement selection, then (54) becomes ungrammatical because here C selects for Neg. The solution to this problem is obligatory  $C \rightarrow T$  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  $\alpha$  is its (non-adjunct) sister in configuration [P,  $\alpha$ ]; then:

### (55) Probe-goal

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

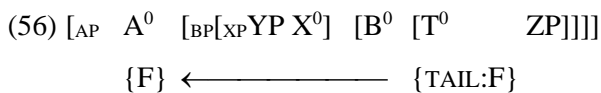
For everything else than criterial features, G must be found from a primitive head to the left of the right edge node. Thus, if [H,  $\alpha$ ] is at the right edge and H is a primitive constituent, feature G is searched from H. If H is complex, it will not be explored (unless G is a criterial feature). 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. But consider again the case of  $C \rightarrow T$ . It is obvious that 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 intervention.

### 3.3.5 Tail-head relation

A tail-head relation is triggered by a lexical feature (TAIL:F<sub>1</sub>,...,F<sub>N</sub>), 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 unambiguous mother-of relation. In the example (56), the tail-feature (TAIL:F) at T<sup>0</sup> can see the feature at the head A<sup>0</sup>, and B<sup>0</sup>, but not at X<sup>0</sup>.



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. Thus, if T has a tail feature (TAIL:F,G), but A only has the feature F, then the tail-head relation fails because there is only partial match; consequently, A must possess both features F and G. For the tail-head relation, it does not matter if T is inside and adjunct or not.

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. 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*.

### 3.3.6 Minimal finite edge

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

## 3.4 Movement reconstruction

### 3.4.1 Introduction

Movement reconstruction is a ‘reflex-like’ operation that is applied to a phrase structure  $\alpha$  and takes place without interruption from the beginning to the end without the possibility of backtracking. From an external point of view, it therefore constitutes ‘one’ step; internally the operation consists of a determined sequence of steps. All movement inside  $\alpha$  is implemented if and only if Merge( $\alpha$ ,  $\beta$ )(Section 2.4). Movement reconstruction is an error recovery procedure.

The operation targets a phrase structure  $\alpha$  and follows a predetermined order: head movement reconstruction  $\rightarrow$  adjunct movement reconstruction  $\rightarrow$  A'/A-movement reconstruction. Head movement reconstruction can take place in one of two ways: (a)  $[\alpha\{\beta\}, \gamma] \rightarrow [[\alpha, \beta], \gamma]$  or  $[\alpha [\gamma \dots \beta \dots]]$ . Option (a) will be selected if  $\alpha$  is primitive, has an affix, and  $[\alpha, \beta]$  satisfies LF-legibility (Section 3.7). In that case no further reconstruction operation will be applied to  $\alpha$ ; otherwise ( $\alpha$  is complex or  $[\alpha, \beta]$  does not satisfy LF-legibility) option (b) is selected (Section 3.4.2), followed by adjunct movement reconstruction (Section 3.4.3) and A'/A-movement reconstruction (Section 3.4.5). As said earlier, reconstruction is not an operation in which various alternatives are tried in some order; it is an all-or-none phenomenon.

### 3.4.2 Head movement reconstruction

Head movement reconstruction travels downward on the right edge of  $\alpha$ , targets primitive constituents at the left that have an affix  $\beta\{\phi\}$  and drops the head/affix  $\phi$  downstream if a suitable position is found. Dropping is implemented by trying to “fit” 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  $\phi$  has no EPP feature and can be selected in that position by a higher head; (ii) it has an EPP feature, has a specifier, and can be selected in that position by a higher head. The left complex sister constitutes an acceptable specifier  $[XP, \phi]$ . These are the basic conditions which capture the notion that heads are 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 that has EPP and is selected properly. If the head has an EPP feature, is selected properly, has no specifier, but occurs in configuration  $[\alpha, \beta]$ ,  $\beta$  being a primitive head without affixes, the solution is accepted without a specifier, because  $\beta$  is a bottom node. 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  $\phi$  can itself be complex. The algorithm will encounter the complex  $\phi\{\gamma\}$  later when it travels downwards from the position it originally extracted  $\phi$ , and applies the same algorithm to  $\gamma$ . A consequence of these rules is that heads must be reconstructed into the first position in which it can be selected; lower positions are never explored.

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  $[\alpha [\phi XP]]$  as a last resort; an unreconstructed head crashes at LF-legibility.

### 3.4.3 Adjunct reconstruction

Adjunct reconstruction begins from the top of the phrase structure  $\alpha$  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: (i) it is complex; (ii) it has not been moved already; (iii) it has a tail set; (iv) floating is not prevented by feature  $[-\text{FLOAT}]$ . Condition (iv) is stipulative and 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.

Suppose  $\alpha$  satisfies conditions (i-iv) above. Then if the tail-head features (Section 3.3.5) are not satisfied in this position (Section 3.4.4),  $\alpha$  must be subject to adjunct reconstructions and is designated as a floater. The functional motivation of this condition is clear: we know that the constituent is not at its canonical position. In addition, (a) if its tail-head features are satisfied but  $\alpha$  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 [ $-\text{SPEC}^*$ ], 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.

After a floater is detected, it will be dropped. Dropping is implemented by first locating the closest finite tense node. 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: (i) tail-head features must be satisfied (Section 3.4.4); (ii) we are not trying to drop into the same projection where we started; (iii) we are not creating new specifiers for a head. 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.

#### 3.4.4 *External tail-head test*

External tail-head test checks if the tail-features of a head (i) can be checked by any of the c-commanding head(s) or if (ii) it is located inside the projection of such head. Tail features are checked in sets: if a c-commanding (i) or containing (ii) 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 being one of them.

#### 3.4.5 *A'/A-reconstruction*

Suppose A'/A-reconstruction is applied to  $\alpha$ . The operation begins from the top of  $\alpha$  and searches downstream for primitive heads at the left or (bottom) right. Three conditions are checked: (1) If the head  $h$  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. (2) If the head  $h$  has the EPP property and has a specifier or several, they are stored into the memory buffer. (3) If the head  $h$  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 (1): fill in the SPEC position. If  $h$  does not have specifiers, a constituent is selected from the memory buffer if and only if either (i)  $h$  has a matching specifier selection feature (e.g.,  $v$  selects for DP-specifier) or (ii)  $h$  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 (ii) is not yet fully implemented, as the generalized EPP mechanism involved (Brattico 2016; Brattico, Chesi, and Suranyi 2019) is not formalized explicitly. The condition, however, checks if phi-features are missing from the head but can be satisfied by merging a DP from the memory buffer. It is a “formal” specifier filling operation. An additional possibility is if the “specifier” of  $h$  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 (2): 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  $h$  is an EPP head: thematic constituents are not reconstructed by this operation. Let us examine the single specifier case first. A specifier is a complex left aunt constituent  $\alpha P$  such that  $[\alpha P [h(+EPP) XP]]$ ,  $\alpha P$  has not been moved already somewhere else. If  $h$  has the generalized EPP feature  $\text{PHI}:0$ ,  $\alpha P$  will undergo A-reconstruction (local successive-cyclicity, Section 3.4.6); otherwise it will be put into memory buffer. The latter option leads possibly into long-distance reconstruction (A'-reconstruction). If  $\alpha P$  has criterial features (which are scanned from it), formal copies of these features are stored to  $h$ . A formal copy of feature  $F$  is denoted by  $uF$ . If  $h$  is a finite head with feature  $\text{FIN}$ , a scope marker feature  $iF$  is created as well. This system means that  $F$  is the original criterial feature,  $uF$  is the formal trigger of movement, and  $iF$  is the semantically interpretable scope marker/operator feature. Lexical redundancy rules and parameters are applied to  $h$  to create a proper lexical item in the language being parsed ( $h$  might have language-specific properties). In addition, the label of  $h$  will be also copied to the new head, implementing the “inverse of feature inheritance.” Finally, if  $h$  has a tail feature set, an adjunct will be created (`create_adjunct`). 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 nonadjuncts, 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  $h$  (`engineer_head_from_specifier`)(57). The latter takes care of the requirement that when  $C$  is created from finite  $T$ ,  $C$  will also have the label  $\text{FIN}$ . If the new head has a tail feature set, an adjunct is created (`create_adjunct`). This operation is required when a relative pronoun creates a relative operator, transforming the resulting phrase into a relative clause adjunct.

$$(57) [\alpha_P \ F \dots] \quad g^0 \quad \beta_P \ h^0$$

$$\longrightarrow \leftarrow$$

Option (3): fill in the complement position from memory (fill\_comp\_from\_memory). A complement for head  $h$  is merged to  $\text{Comp}_h$  from the memory buffer if and only if (i)  $h$  is a primitive head, (ii)  $h$  does not have a complement or  $h$  has a complement that does not match with its complement selection, and (iii)  $h$  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 3.4.7).

### 3.4.6 A-reconstruction

A-reconstruction is an operation in which a DP makes a local spec-to-spec movement, i.e.  $[\text{DP}_1 [\alpha \text{ } \_1 \beta]]$ .

The operation is implemented if and only if  $\alpha$  has the generalized EPP property: we assume that DP has been moved locally to satisfy this feature.

### 3.4.7 Extraposition as a last resort

Extraposition is a last resort operation that will be applied to a left branch  $\alpha$  if and only if after reconstruction all movement  $\alpha$  still does not pass LF-legibility. The operation checks if the structure  $\alpha$  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  $[K \text{ HP}]$  or  $[K \langle \text{HP} \rangle]$ . 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  $\text{HP} = [H \text{ XP}]$  such that (i) HP is adjoinable in principle (Brattico 2012)<sup>5</sup> and either (i.a) there is a head  $K$  such that  $[K \text{ HP}]$  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  $[KP \text{ HP}]$ . 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 3.4.8). This will transform  $[K \text{ HP}]$  or  $[KP \text{ HP}]$  into  $[K \langle \text{HP} \rangle]$  or  $[KP \langle \text{HP} \rangle]$ , respectively. Only the most bottom constituent that satisfies these conditions will be promoted; if this does not work, and  $\alpha$  is still broken, the model assumes that  $\alpha$  cannot be fixed.

---

<sup>5</sup> 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:

(58) John+    gave+    the book+    to    Mary  
       [John    [T        [DP[ the book [P    Mary]]]  
                       SPEC    P    COMP

This interpretation is broken in several ways. 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 (59).

(59) [John [gave [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*’.

### 3.4.8 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.” The operation is part of reconstruction; decisions concerning what will be an adjunct and non-adjunct cannot be made in tandem with consuming words from the input. 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, so that if H is targeted for promotion in a configuration [<sub>HP</sub> 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. In the current implementation (which will possibly require revision) it is assumed that if HP is in a canonical position as determined by its tail-head features and if it has the (!SPEC:\*) feature, then whatever constituent contains HP will be taken with



HP, thus it will contain the mandatory specifier. If there is no specifier, then the operation can still be done as the expression is crash either way. If HP is not in its canonical position, on the other hand, then the specifier is carried along if and only if (i) H does not have ( $-\text{SPEC:}^*$ ) feature explicitly banning a specifier and (ii) the label of the specifier L is not rejected by H ( $-\text{SPEC:L}$ ). To see why this is relevant, consider the relative clause. A relative clause is headed by a relative operator at C. When the relative clause is promoted into an adjunct, we must promote the relative pronoun at Spec,CP as well, hence we promote the whole CP. The current implementation will not count specifiers, it only promotes the head, its complement and (given the above condition) the constituent that hosts HP.

### 3.5 Agreement and control

Agreement reconstruction is attempted after all movement has been reconstructed. The operation goes downstream from  $\alpha$  and targets any primitive head to the left that has unvalued  $\phi$ -set ( $\phi\_$ ) and requires valuation (does not have feature  $-\text{VAL}$ ). Such heads H attempt to *acquire*  $\phi$ -features. Acquisition of  $\phi$ -features consists of two steps: (i) acquisition from  $\phi$ -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  $\phi$ -features from the first such element. Operation (ii) picks up the first generalized specifier (adjunct or non-adjunct) with label D and obtains  $\phi$ -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  $\phi$ -features are then *valued* to the head. Denote a  $\phi$ -feature of the type T with value V as  $[\text{PHI:T:V}]$ . An acquired  $\phi$ -feature  $\{\text{PHI:T:v}\}$  can only be valued at H if (i) H contains  $\{\text{PHI:T:}_\_\}$  and (ii) no conflicting feature  $\{\text{PHI:T:v}'\}$  exists at H. Condition (ii) leads into  *$\phi$ -feature conflict* which, when present, crashes the derivation at LF. Thus, condition (ii) does not terminate the operation, but is illegitimate at LF.

### 3.6 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 space need 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 corresponds 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),  $\phi$ -features (e.g., PHI:NUM:SG) and others. The number and type of lexical features is not restricted by the model.

### 3.7 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, as follows. 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). *Criterion 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: Aquila & Celik.
- Brattico, Pauli. 2019. "Finnish Word Order and Morphosyntax." *Manuscript*.
- Brattico, Pauli and Cristiano Chesi. 2019. "A Top-down Approach to Pied-Piping." *Manuscript*.
- 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*. MIT Press.
- 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, C. 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.