Computational implementation of a top-down parser-grammar

Pauli Brattico

IUSS, Pavia

Abstract

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

# 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 such questions 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: linguistic representations internalized by the speaker. One might then want to take a further step and examine also linguistic performance, use of that knowledge. How do speakers and hearers put that knowledge in use in real-time communication? What kind of mistakes do they make? The latter inquiry presupposes the former: a theory of language use requires an explicit or implicit theory of the language – its lexicon, rules and categories – whose use is being studied. Competence, on the other hand, can be studied without taking performance into account.

It does not follow from this otherwise sound methodological principle that the facts pertinent to linguistic performance must be irrelevant to the theory of competence. Whether performance should be taken into account in a theory of competence can only be answered by empirical inquiry. One strong (and hence interesting) form of a theory which assumes that performance should be taken into account in 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 inalienable part of the theory of competence. Specifically, the main hypothesis pursued in this study is that the properties of the PF-interface, when looked from the perspective of language comprehension, matter for the theory competence.

# The framework

## Merge

Linguistic input is first received by the hearer in the form of sensory stimulus. That input can be thought abstractly as a one-dimensional, linear string α, β, … of phonological words. We assume that in order to understand what the sentence means, the human parser must create a set of abstract syntactic interpretations for the input string received by 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 incrementally into a hierarchical representation. For example, if the input consists of two words α, β, Merge will yield [α, β](1).

1. John + sleeps.  
    ↓ ↓  
    [John, sleeps]

The assumption that this process is incremental 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*, then it will be merged directly with (1). There are three possible attachment sites, shown in (2), all which correspond to different hierarchical relations between the words.

1. a. [[John *furiously*], sleeps] b. [[John, sleeps] *furiously*] c. [John [sleeps *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 subject. 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 (2)a. Therefore, this alternative can be ruled by the fact that the result is ungrammatical and that it is not consistent with the word order in the input. But what *is* consistent with the linear ordering? If the default linearization proceeds recursively in a top-down left-right order, then each word in the input must be merged to the “right edge” of the phrase structure, “right edge” referring to the top node or to any right daughter node, recursively. This would mean either (b) or (c) in (2). (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, but which one?

There are many situations in which the correct answer is not known, for example, if the sentence is ambiguous, or can be known but is unknown at the point when the word is consumed. This follow from the truism that in an incremental parsing process, decisions must be made concerning an incoming word without knowing what the remaining words are going to be. We can experience this situation when reading a garden-path sentence: a wrong decision is made but the error is recognized when reaching the end (as in *The horse raced past the barn fell*). It must be possible to backtrack and re-evaluate a parsing decision made earlier. To incorporate this mechanism, let us assume that all legitimate merge sites are orderedand that these orderings generate a recursive search space that can used to backtrack. The situation after consuming the word *furiously* will thus look as follows, assuming an arbitrary ranking:

1. ~~a. [[John~~ *~~furiously~~*~~], sleeps]~~ b. 1. [[John, sleeps] *furiously*] c. 2. [John [sleeps *furiously*]]

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). Let us assume that the parser-grammar can establish such ranking and is able to explore the whole derivational space recursively.

Both solutions (a) and (c) are “countercyclic”: they extent the phrase structure at its right edge, not at the highest node. 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,” whether it is up or down.

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 (1) and (2)c, repeated here as (4).

1. John sleeps furiously  
    ↓ ↓ ↓  
    [John, sleeps] → [John [sleeps furiously]]

During the first stage, the words *John* and *sleeps* are sisters inside the same constituent [John, sleeps]. If the adverb is merged as the sister of *sleeps*, this no longer holds: no *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 (5).

1. [John [[sleeps furiously] γ]

This property of the Phillips architecture has several 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 (5), 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 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 2.3. For present purposes, it is important to keep in mind that in the Phillips architecture constituency relations established at point t1 do not necessarily hold at a later point t2.

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. Perhaps Merge can operate in a production “mode” and in comprehension “mode”. The weaker hypothesis is less interesting than the strong one, and possibly wrong in being less parsimonious, but it allows me 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.

## 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 (6).

1. John admires Mary  
    ↓ ↓ ↓  
    [John [admires Mary]]

There is much linguistic evidence that this derivation generates the correct hierarchical relations between the three words. The verb and the direct object form a constituent that is merged with the subject. We can imagine that this hierarchical configuration is interpretable at the LF-interface, with the usual thematic/event-based semantics: Mary will be the patient of the event of admiring, whereas John will be the agent. If we change the positions of the arguments, the interpretation changes. But the fact that the verb *admire*, unlike *sleep*, can take a DP argument as its complement must be determined somewhere. Let us assume that such facts are part of the lexical items: *admire*, but not *sleep*, has a *lexical selection feature* [!comp:d] which says that it is compatible with, and in fact requires, a DP-complement. The idea has been explored by (Chesi 2004) 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 the ranking: when *Mary* is consumed, the operation checks if any given merge site allows the operation. In the example (7), the test is passed: the label of the selecting item matches with the label of the new arrival.

1. John admires Mary  
    ↓ ↓ ↓  
    [John [admires Mary]]  
    [comp:d] d ✓ Merge site accepted.

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.

Let us return to the example with *furiously*. What might be the lexical features that are associated with this item? The issue depends, of course, on the specifics of assumed theory of competence, but let us assume something for the sake of the example. There are three options in (7): (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 [-comp:adv]. We are left with the two options. Let us examine the two output conditions more carefully (8).

1. a. [[S John [admires Mary]] furiously]  
   b. [John [VP admires Mary] furiously ]]

Completely 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* (left phrases) it is allowed or required to have. I will call such features *specifier selection features*. A feature [spec:s] favors solution (a), whereas [spec:v] favors solution (b). Analogously with the complement selection features, we use “!” to indicate that the selection is mandatory. If the adverbial has both feature or neither, then the selection is arbitrary. More generally, suppose the item to be is merged is α; the specifier features of α will tell what type of complex phrases LP α can be merged with. The term “specifier” is here used slightly differently from the standard usage, but it is not misleading because in most cases the selected phrase [LP, α] will end up constituting a specifier of the element α that is merged.[[1]](#footnote-1) An accurate but also cumbersome name would be “left phrase selection feature.”

## Phases and left branches

Let us consider next the derivation of a slightly more complex clause (9).

1. 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 the DP. This follows from the assumption that merge is always to the right edge, following Phillips. It now follows that all left branches become *phases* in the sense of (Chomsky 2000, 2001). This “left branch phase condition” was argued specifically by (Brattico and Chesi 2019a).

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

If no future operation is able to affect a left branch, as assumed in (10), 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 by Merge. Furthermore, if, after all operations have been done, the left branch fragment is ungrammatical or uninterpretable, then the original merge operation that created the left branch phase must be cancelled. This is because no future operation will be able to fix the issue. 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 unusable.

## Labeling

Suppose we reverse the arguments (9)and derive (11).

1. 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 itself only 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 (12) was used. The intuitive function of the algorithm is to “search” for the closest possible primitive head from the phrase.

1. 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 (12) to it; otherwise,  
    d. apply (12) to the left constituent.

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*. Notice that, according to this definition, a constituent that has only the left or right constituent will still be primitive (this becomes relevant later). Conditions (12)c-d mean that labeling – and hence selection – ignore adjuncts. This will be a defining property of what adjuncts are. A situation in which both the left and right constituents were adjuncts was ruled out.

Consider again the derivation of (1), repeated here as (13).

1. John + sleeps.  
    ↓ ↓  
    [John, sleeps]

If *John* is a primitive constituent having no left or right daughters, as seems to be the case, the then labeling algorithm will label [*John sleeps*] as a DP. Its structure is [D V], thus the primitive left constituent D will also be its label. This is a wrong result. There are at least three ways to solve this problem. One solution is to reconsider the labeling algorithm. That seems implausible: (12) captures what looks to be a general property of language, thus this alternative would require us to treat (13) and other similar examples as exceptions. They are not exceptions, however: there is nothing anomalous about (13). The representation should come out as a VP, with the proper name constituting the argument of the verb. Thus, the proposer 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 would be a “complex constituent” despite having no constituent is a contradiction in terms. 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 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 I find it 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; and even worse, it could make every item ambiguous for the parser.

For these reasons, I kept the definition of labeling as provided above and assumed that *John* is a complex constituent despite of appearing 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 comes from the lexicon/morphological parser. The structure of (13) is therefore (14), with “D” and “N” coming from the lexicon/morphology.

1. John + sleeps.  
    ↓ ↓  
    D N sleeps  
    ↓ ↓ ↓  
    [VP[DPD N] sleeps]

It is important to notice that the labeling algorithm (12) 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 (14), 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 (15).

1. [VP[DP D N] [ V0 [DP D N]]

## Adjunct attachment

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 2.4 ignores adjuncts. The result is that the label of (16) is V: while analyzing the higher VP shell, the search algorithm does not enter inside the adverb phrase; the lower VP is penetrated instead (12)c-d.

1. H0 [VP John [VP[VP admires Mary] 〈AdvP furiously〉]]

The reasoning applies automatically to selection by the higher head H: if H has a complement selection feature [comp:v], it will be satisfied by (16). Consider (17).

1. John [sleeps 〈AdvP furiously〉]

The adverb(ial) constitutes the sister of the verbal head V0 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. Again, *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 (17) is [VP[*John*] *sleeps*]. The reason adjuncts must constitute geometrical parts of the phrase structure is because they can be targeted by several syntactic operations, such as movement and case assignment (Agree).

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 a label or feature. The linking relation is established by means of an ‘inverse probe-goal relation’ that I call *tail-head relation*. For example, a VP-adverbial is linked with V, a TP-adverbial is linked with T, a CP-adverbial is linked with C, and so on. 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 (18).

1. *Condition on tail-head dependencies* A tail feature F of head H [tail:f] can be checked if either (a) or (b):  
    a. H occurs inside a projection whose head has F.  
    b. H is c-commanded by a head whose head has F.

Of these conditions, (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 in Section 3). A VP-adverbial will be reconstructed inside a VP, and so on. I will keep (ii); but the matter becomes important only when we address the ‘free word order’ property. A tail-head relation is checked by (18).

If an adverbial/head does not satisfy a tail feature, it will be reconstructed into a position in which it does. This operation will be discussed in Section 3.4. But 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 some ‘stage’ in the derivation. An important property of the rule (18) and many similar rules is that each constituent must satisfy such rules only once, or in one position. Ignoring the ordering of operations in the derivation, if C1, . . .Cn is a set 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 (18), is that the last 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 the parser-grammar framework, the representation that is generated last is often or always the one that is 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) after ignored and only the last position will be checked. This is not always the case, however. The EPP property (Section 2.6) 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.

## 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 be in its canonical position, and hence the existence of this feature on a head triggers A´/A movement reconstruction (Section 3.2). This constitutes a sufficient (but not necessary) feature for reconstruction; see 3.2. Language uses phrasal movement, and hence an unselective specifier feature, to represent a null head (such as C) at the PF-interface (Section 3.1).

## Move

### Why movement: entanglement between word order and morphosyntax

The reason linguistic constituents can be displaced from their canonical positions has remained a profound mystery. Once we adopt the comprehension perspective, the mystery disappears. A moved *wh*-phrase represents the null head C(*wh*) at the PF-interface. A feature that cannot be represented at PF by means of a phonologically interpretable feature can be represented by moving a phrase to it vicinity (SPEC position), thus, to mark its existence at the PF-interface. This makes it possible to infer the existence of a grammatical head from word order. The opposite is true as well. In a free word order language such as Finnish, it is possible to infer word order from rich morphosyntax. The two-way relation represents an “entanglement” between order and morphology/phonology (Brattico 2019). We can think of the two phonological properties, order and morphosyntax, as being the two side of the same coin.

### A´/A 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, the latter which then constitutes a canonical position in terms of the finished phrase structure. For example, regular referential or quantificational arguments must occur inside the verb phrase at the LF-interface in order to get thematic roles. The problem for the parser-grammar is to determine when a phrase or a head in the input must be reconstructed into a canonical LF-position. There are at least two necessary conditions. One condition is the occurrence of a lexical element, such as the finite tense, that has the [spec:\*] feature and has a phrase at its specifier position. The position is not thematic, hence the parser can infer that the phrase must be reconstructed to a canonical LF-position in which it is selected thematically. Another condition is the occurrence of a criterial feature. Criterial features are used to infer the existence of a licensing head.

### A´-reconstruction and criterial features

Consider (19).

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

The clause begins with two arguments, the first which hosts a criterial *wh*-feature. To compute properties of (19) correctly the parser-grammar must infer the existence of a operator head C(*wh*) on the basis of the moved phrase and the criterial feature (20). This is, in fact, the whole function of movement in this particular case: to signal the force and scope of the interrogative clause.

1. 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 step is the recognition that a head is missing. That can be inferred (again, in this particular example) from the existence of what looks to be the two specifiers of the T/fin head. If the head is not missing, as in the case of English *did* support, nothing needs to be done. Suppose the head is missing, as is the case with Finnish. Then the parser-grammar will generate the required head to the structure. The head will be generated into a position in which the phrase will be its unambiguous specifier, as shown in (20). The next step is to generate or infer the label of the head. This information is obtained from the criterial feature. Thus, if the moved phrase contains a *wh*-feature, that feature will be ‘copied’ from the phrase to the head, as shown in (20). If the phrase had another feature, such as contrastive focus, then that feature would be copied. This allows the parser-grammar to infer both the existence and the nature of the phonologically null head. This ‘copying’ operation resembles Agree of the standard bottom up theory; I will discus it later in detail.

The phrase must then be reconstructed back to its canonical LF-position. The intuition is that it will be stored into a syntactic memory buffer and reconstructed into an empty position downstream. The mechanism was first proposed and developed in (Chesi 2004) in connection with the top-down theory. To implement the system formally, we need to specific two further facts: how are empty positions detected and when does reconstruction occur? When it comes to the former problem, I will adopt Chesi’s original idea that empty positions are determined by lexical selection features. If a head is encountered that selects for a label L, but the selected constituent is not present, memory buffer is consulted to determine if a suitable phrase is found there; if it is, the phrase will be merged from the memory buffer to the empty position. In the example (21), the interrogative pronoun will be first stored into the memory buffer and then reconstructed into the complement of the transitive verb once the parser-grammar detects that a required argument is missing (i.e. its [!comp:d] feature is not satisfied).

1. Who does John admire \_\_?  
    ⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯→

I will assume, for reasons that become clear later, that deconstruction is a form of copying. Thus, the interrogative pronoun is copied from the criterial position into the memory buffer, from where it is copied to the canonical LF-position (“\_\_”).

Chesi assumes that phrases are stored into the memory buffer and retrieved into canonical LF-positions in tandem by consuming words. This is possible only if the intended phrase structure is known in advance; it is not possible in the context of parsing, in which we are working with PF-interface information (linear string of words). I will assume that movement is reconstructed separately for each left phrase (phase) and then for the whole structure once all words have been consumed (Brattico and Chesi 2019a). Formally, movement is reconstructed inside XP during Merge(XP, α). There are several reasons why this assumption is necessary. The most obvious reason is that it is impossible to assess the grammaticality of XP unless we attempt to reconstruct movement inside XP. Leaving movement undone would leave several thematic positions empty, and the construction would evaluate as ungrammatical/uninterpretable.

### A-reconstruction and EPP

Another sufficient condition for phrasal movement is the occurrence of a phrase at the specifier position of a head that has the [spec:\*] feature (=EPP in the standard theory). The mechanism is the same as that of the A´-reconstruction (Section 3.2.1), but the grammatical features involved differ. The ‘criterial features’ of this operation are the ϕ-features and not, say, *wh*-features, as shown in (22).

1. Me ihaile-mme Merjaa. (Finnish)  
    **1pl**.nom admire-**1pl** Merja.par  
    ’We admired Merja.par.’

The finite tense node agrees in ϕ-features with a phrase that typically (but not necessarily) moves to its specifier position. The copying/agreeing operation is the same as it was in the case of criterial features: features from the moved phrase are copied to the head (22). These morphosyntactic features again substitute for order: in languages/constructions with rich agreement, the grammatical subject can be null or get displaced into a (virtually random) position in the clause (Brattico 2016).

### Head reconstruction

Many heads occur in noncanonical positions in the input string. In Finnish, for example, verb-initial clauses are ungrammatical due to the [spec:\*] feature at T/fin, unless a head has been moved to the C-head either to generate a verum focus interpretation (corrective focus scoping over the whole sentence) or to create some other interpretation associated with a criterial feature present in the head. In the example (23), a head in an embedded finite clause has been suffixed with the yes/no clitic -kO and then fronted.

1. Nukkua-ko Pekka ajatteli että hänen pitää \_?  
    sleep-Q Pekka thought that he must   
    ‘Was it sleeping that Pekka thought that he must do?’

Lexical and morphological parser first 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 Tin (-*a*-)and V (*nukku*-). Morphology extracts this information from the phonological word and feeds it to syntax (24). Symbol “#” indicates that there is no word boundary between the morphemes.

1. C(-kO) + #Tinf + #V + Pekka + ajatteli + että + hänen + pitää   
    C T V Pekka thought that he must

Morphology has no access to syntax; instead, it decomposes phonological words and feeds them into syntax in a *linear order*, one morpheme at a time. The individual heads are then 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 cliticalization. Thus, if syntax receives a word-internal morpheme, it will be merged to the *right edge of the previous morpheme*. Technically the constituent is [α ∅ β] but I will denote it as α{β}. Notice that by “defining complex” constituent as one that has both the left and right constituent, [α ∅ β] comes out as “primitive constituent” and can therefore constituent the head of a projection. The linear sequence C-T-V becomes C{T{V}}(or [C0 ∅ [T0 ∅ V0]]), and this is what gets merged (25).

1. C{Tinf{V}} Pekka ajatteli että hänen pitää \_\_  
    Pekka thought that he must

Head reconstruction will return the constituents of the complex head into their canonical positions when the whole phrase is merged as a left constituent. This is done by targeting the highest head and finding the closest position in which it can be selected and in which it does not violate local selection rules. In the example (25), this will be the Tinf{V}. The closest possible position for Tinf is the empty position inside the embedded clause. V will be extracted from the complex head in the same way and reconstructed into the complement position of Tinf. The same operation will extract D and N from a proper name such as *John*: D{N} → [D N].

1. C{Tinf{V}} Pekka ajatteli että hänen [pitää [Tinf{V}V]].  
    ⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯→ ⎯⎯→

The operation reverse-engineers head movement in production. Right constituents in the absence of the left constituent provide a notion of “intermediate” constituent. Consider pronouns, clitics and agreement. A full pronoun has the status of [D N] as it fills in an argument position and does not project. Minimally, it contains a ϕ-set. A pronoun can be “cliticized” to the right edge of a word by merging it to the right constituent. It then behaves like a pronominal head (a clitic). An additional alternative is to copy the ϕ-features to the head as a feature (27). A right constituent ϕ will be extracted by head reconstruction into at ϕ-head (an Agr head).

1. a. [D N] = [ϕ ϕ] = pronoun  
   b. T{ϕ} = [T0 ∅ ϕ] = clitic  
   c. T[ϕ] = agreement

### Adjunct reconstruction

Consider the pair of expressions in (28) and their canonical derivations.

1. 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, whereas (b) is incorrect. The thematic roles are identical in both examples. 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). 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 in (29).

1. Case features must establish local tail-head relations such that  
    a. [nom] is checked by +fin,  
    b. [acc] is checked by v (asp),  
    c. [gen] is checked by −fin  
    d. [par] is checked by −phi.

The symbol “−phi” refers to a head that never exhibits ϕ-agreement. If the condition is not checked by the position of an argument in the input, then the argument is treated as a an adjunct and reconstruction into a position in which (29) is satisfied. In this way, the inversed subject and object can find their ways to the canonical LF-positions (30). Notice that because the grammatical subject *Pekka* is promoted into adjunct, it no longer constitutes the complement of the verb; the partitive-marked direct object does.

1. [〈Merjaa〉2 T/fin [ \_\_1 [käski [meidän [ihailla [ \_\_2 〈Pekka〉1]]]]  
    +fin ←⎯→ [nom] −phi ←⎯→ [par]  
    Merja.par asked we.gen to.admire Pekka  
    ’Pekka asked us to admire Merja.’

Rich case suffixes function as ‘adverbializers’: the allow an argument to find its way into a canonical LF-position irrespective of its position in the word order. The same mechanism will reconstruct adverbials: a VP-adverbial will find its way into the vicinity of the V-head, a TP-adverbial into the vicinity of T, and so on.

## Ordering of operations

The catalog of computational operations required to construct a phrase structure from a linear string of words must itself be ordered in some way. There seems to be only one way to order them. Both A´/A-reconstruction and adjunct reconstruction presuppose head reconstruction, because it is only the presence of heads and their lexical features that can guide A´/A- and adjunct reconstruction. The former relies on EPP features and empty positions, whereas the latter relies on the presence of functional heads. Furthermore A´/A-reconstruction relies on adjunct reconstruction: empty positions cannot be recognized correctly unless orphan constituents that might be hiding somewhere are first returned to their canonical positions. All three movement operations presuppose merge from the input, and the whole sequence presuppose that the merge has constructed a left branch (31).

1. Derive α → Merge [α, β] → head reconstruct α → adjunct reconstruct α → A`/A reconstruct α

## Lexicon and morphology

Most phonological words enter the system a polymorphemic units that might be further associated with inflectional features. The morphological component is responsible for decomposing phonological words into these components. A table-look up dictionary matches phonological words directly with their morphological decompositions. This corresponds with an automatized pattern recognition procedure. The decomposition consists of a linear string of morphemes *m*1#...#*m*n that are separated and inserted into the linear input individually (32). Notice the reversed order.

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

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 ϕ-features, and occurs above the finite tense node in Finnish (Holmberg et al. 1993). Its properties differ from the English negation *not*. Some of these properties are so idiosyncratic that they must be part of the language-specific lexicon. One such property could be the fact that the negation selects T as a complement, which must be stated in the language-specific lexicon to prevent the same rule from applying to the English *not*. It is assumed that language-specific features *override* features emerging from the two remaining sources if there is a conflict.

Another source of lexical features comes from a set of universal redundancy rules. For example, the fact that the small verb v selects for V need not be listed separately in connection with each transitive verb. This fact emerges from a list of universal redundancy rules which are stored as feature implications. 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. Redundancy rules are not ‘lexical’; instead, they describe lawful connections between features, a type of mini-grammar. The rule ‘[cat:v] → [comp:v]’, for example, provides that the functional element v is ungrammatical/uninterpretable without a verbal complement.

A third source of lexical features comes from a curious set of parametric rules. For example, it turns out that in Finnish most heads that have the [spec:\*] feature also exhibit ϕ-agreement in some context (e.g., prepositions, noun heads), whereas the same functional heads in English have neither (Brattico and Chesi 2019b). 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.

When a lexical item is retrieved, its feature content is first fetched from the language specific lexicon and is then processed through the redundancy rules and parameters. If there is a conflict, the language-specific lexicon wins.

Lexical features constitute an unstructured set, but the features themselves can constitute ‘type:value’ pairs. For example, labels are provides as values (n, v, a, …) of the type cat(egory).

## 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 a [!spec:d] specific selection feature. 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 a more ‘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, if any. 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 contains 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. 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 (33). It therefore tails a feature pair cat:v, sem:directional. There is no limit on the number of features that a verb can posses and a prepositional argument that tail.

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

# Full formalization and the human language parser

## Introduction

The framework delineated in Section 2 consists of assumptions that follow, some of them by virtual necessity, from the assumption that parsing is incremental and uses the core computational operations such as Merge. A full formalization and computational implementation, on the other hand, requires a set of conjectures, working hypotheses and further details that can be demonstrated to be “functional” by means of computer simulation but are less inevitable and thus subject to alternative formalizations and approaches. In addition, there are several ideas that are part of the overall framework but not implemented, such as various operation that have to do with discourse semantics and communicative pragmatics. These ideas are collected into this section. The section is organized in a top-down fashion, discussing more significant and broader issues first, followed by the discussion of the details. The ‘algorithm’ delineated here is assumed to represent a realistic description of the human parser, not a computer simulation in the sense of “automatization”; matters related to pure automatization and the Python implementation are discussed in Section 4. That section is meant as a guide for understanding the implementation.

## Overall structure of the algorithm

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

A close up of a map

Description automatically generated

*Figure 1. Structure of the language comprehension algorithm.*

The individual components are commented in the subsections below.

## Morphological and lexical processing

## Application of lexical selection

## Reconstruction of movement

## LF-legibility

# Computational implementation

## Coding conventions

### Variable names

The following naming conventions were used.

ps = phrase structure of any kind.

ps\_iterator = a pointer to a (typically) right edge of a phrase structure. Relevant elements are then sisters (heads and phrases).

h = grammatical head or primitive constituent.

1. Consider the tense node T in English. Finite T will have a specifier selection feature [!spec:d] which causes it to be merged with a DP (if any). This generates [DP, T]. If the next item is the verb, it will be merged to the complement of T. This will create [DP [T V]] which will, if the derivation continues as expected, leave us with a representation in which the DP will constitute the specifier of the finite T node. [↑](#footnote-ref-1)