Computational generative grammar and complexity

Pauli Brattico

2024

Abstract: Advanced biolinguistic theories have an adverse complexity profile in which the number of calculations required to verify them against nontrivial datasets increases exponentially as a function of data complexity. The task implies millions of calculation steps even for regular sentences and soon becomes infeasible for all traditional paper-and-pencil methods. In this article we propose a computational solution to this problem. We formalize a variant of the bottom-up minimalist grammar, embed the result inside a computational infrastructure which automatizes linguistic reasoning and then consider the use of this construct in the development, justification, and testing of linguistic theories and hypotheses in the manner originally proposed by Chomsky (1957). We conclude by proposing a more general methodological principle designed to increase precision in linguistic theorizing.

1 Introduction

Most grammars are combinatorial: they create linguistic expressions by combining primitive parts ("words") into larger units ("phrases").[1] The grammar is said to be observationally adequate if it generates attested expressions and only them. The combinatorial nature of grammatical theories creates a problem, however, when the deductive chains from initial lexical selections to whole

[1] This document is a revised version of the original written during Spring 2024 to accompany the Python scripts available at https://github.com/pajubrat/Templates and to explain the motivation behind the approach. The script exemplified in this document is **template2.py**. The rest are explained briefly in Appendix B and in more detail in future work, in preparation. The scripts are specifically written as simple and convenient entry-points for further development rather than fully specified theories with detailed and controversial assumptions.

expressions become so protracted and long as to be virtually impossible to construct and verify by any paper-and-pencil methodology.[2]

We propose a computational solution to this problem. The solution is a general computational function we call *derivational search function* that automatizes all inferential processes involved in demonstrating that a given data follows from the grammatical theory or hypothesis and as such eliminates the complexity barrier from linguistic theorizing. The argument is organized as follows. Section 2 introduces the derivational search function and its implementation. Several auxiliary issues together with simplified examples are addressed. Sections 3–5 introduce more specific computational components (e.g., lexicon, subcategorization, Merge, Move, Agree, adjunctions, labelling and others) to test the methodology in a more realistic setting. Sections 6–7 contain the main conclusions and discussion of the results. Specifically, we show that the number of calculations involved is too large to be handled by any manual method, hence the computational framework. The source code is available in the source code repository (see Appendix B and footnote 1) and is commented only occasionally in this document.

Certain limitations apply to what we can hope to accomplish. Our interest is strictly limited to the problem of justification: given some dataset D, how can we establish with certainty (that is, deductively) that it follows from a grammar or hypothesis T? Here both D and T can be anything as long as they are defined unambiguously. We are not interested in any particular grammatical theory, hypothesis or formalization, though we will formalize one particular class of grammars in order to

[2] Stabler (2011) considers the range of expressions derivable from a particular class of minimalist grammars and observes that "[t]rying all the possibilities is quite tedious by hand, but is much easier when the process is automated" (p. 634). How tedious? The number of calculations increases exponentially as a function of the size of the initial lexical set. To get a rough idea of what this means in a realistic linguistic context, calculating that the properties of an expression such as *the man believes that the dog barked* follow from a fairly standard grammatical theory requires approximately 5 million calculation steps. It is not sufficient that these calculations are performed once; they must be executed each time the theory is changed, and in the context of a realistic research program they must be executed against several expressions that bear on the hypothesis under consideration.

demonstrate how the approach works and to examine some of its properties, especially those pertaining to complexity.[3] In addition, we are not interested in any particular data, dataset or empirical phenomenon; rather, we formulate the framework in such a way that it can be applied to any type of linguistic data. We are also not interested in deducing theorems that concern formal properties of classes of grammars introduced here or elsewhere, or of any formal systems, but rather deducing *data*, much like physicists are principally interested in equations which can be used to deduce empirical patterns discovered by experiments and observation. Hence the approach is concrete and practical insofar as the underlining material – linguistic observations – presents to us as concrete objects.[4] Finally, this paper is neither about engineering (we assume basic knowledge of Python) nor about parsing, both important but separate topics.

2 Derivational search function

The main issue we address in this paper and in the computational script that underpins everything is how to verify with complete certainty that some data (the phenomenon of interest) follows from a theory or hypothesis (any theory or hypothesis). That is, how linguistic hypothesis and theories can be *justified* when justification is understood in the objective sense assumed in the advanced sciences such as physics.

To do this we need a function that explores all logical implications of the theory or hypothesis of interest and then compares the output with empirical observations. Let us assume that there is a syntactic working memory (sWM) which contains all syntactic phrase structure objects (henceforth,

[3] Thus we will not contribute anything new to the existing literature on formalizing grammars, minimalism in particular (e.g., Stabler, 1997, 2011; Graf, 2013; Collins & Stabler, 2016). While we present a formal grammar by creating a Python implementation for one cluster of ideas in order to test something, we do not claim that the formalization is superior to anything else, novel or even plausible. In fact, we assume the opposite, namely that questions of plausibility or correctness (that is, justification) can only be resolved by matching the theory with nontrivial datasets.

[4] For a discussion of abstract properties of classes of grammars, especially in relation to minimalism, see (Salvati, 2011; Stabler, 2011; Collins & Stabler, 2016) and the literature cited therein. This is an important but completely different topic.

syntactic objects) currently under active consideration (e.g., Collins & Stabler, 2016). The derivation begins by populating the sWM with zero-level syntactic objects. The initial set of zero-level objects will be called *numeration*. For example, an expression such as *the dog bites the man* is derived from an initial numeration {*the, dog, bites, the, man*} by merging the elements together in some well-defined order.[5] Let us posit a recursive *derivational search function* that takes the contents of the syntactic working memory as input and applies all operations of the grammar to all syntactic objects in sWM in a well-ordered sequence, updates the contents of the sWM after each operation, and calls the same function recursively with the updated sWM. The derivation ends if only one object is left, after which the result is evaluated and, if it passes as a well-formed output, linearized into a sentence accepted and enumerated by the grammar. In pseudocode:

(1) *Derivational search function*

```
a. Assume a set sWM of syntactic objects as input;
b. if there is only one syntactic object, evaluate and print it out, otherwise
c.1 for each grammatical operation $O_n$ with $n$ arguments:
    c.2 for each $n$-tuple SO = $(X_1, . . . , X_n)$ of items in sWM:
        c.2.1 apply O(SO) = $\alpha$;
            c.2.2 create updated sWM* where $\alpha$ replaces $X_1, . . . , X_n$;
            c.2.3 recurse into (1)a with sWM*.
```

Notice that the derivational search function is not an empirical theory, nor does it represent computational processes executed by the human language faculty. It can be thought of as

[5] We do not reject the possibility of feeding the numeration with complex syntactic objects such as phases or idiom constructions, although this option will not be considered in this article. Moreover, the notion of primitive syntactic object is neither trivial nor easy to characterize, see Section 3. Another possibility is to populate the numeration with linguistic features which are then bundled into lexical items by further combinatorial rules. This would make no difference to what we say below.

"simulating a linguist" attempting to check the correctness of the theory against empirical data. The empirical theory will be embedded into the derivational search function by the grammatical operations O (step c.1) and by the final well-formedness evaluation of the output (step b).

An additional property of (1) is that it is formulated in such a way that it can be applied to almost any grammar or hypothesis: all we require is that the grammar/hypothesis is formulated in terms of some grammatical operations (c.1) and/or some well-formedness conditions on the outputs (b). Operations O are required so that we can at the very least build some syntactic objects, and the well-formedness conditions impose further conditions, if any, on the finished products. It is hard to come up with anything more general. Something must be built from some parts, and the results must be evaluated in some manner. Specifically, we do not require that the operations O are restricted in any way (they may apply freely) or that there are actual well-formedness conditions on the outputs; all we assume is that they may exist as envisioned by the linguist. In addition, the initial set of syntactic objects can be anything (lexical features, lexical items, complex phrases). The actual computational implementation of (1) follows the same logic and refers to the syntactic operations and well-formedness conditions that are defined outside of the derivational search function itself (see appendix A).

Although (1) is derivational and refers to the explicit build-up of syntactic objects, it does not matter if the grammar is. A *representational* grammar captures grammaticality by positing well-formedness conditions, such as the "interface legibility conditions" of the standard minimalist theory, that apply to complete grammatical representations. The derivational search function then constructs the representations by applying the rules of the grammar to the lexical base more or less freely and tests the outputs against data. A *derivational* grammar differs from the representation grammar in that the construction process itself, and not just the output, is subject to grammatical laws and therefore also to empirical evaluation. For example, some minimalist grammars make empirical claims on the basis of the computational resources consumed by the derivations. From the point of view of the derivational search function the difference between representation and derivational explanations is relatively inconsequential: both theories generate grammatical objects by rules and in both cases the output is compared with observation. If the grammar is derivational, then some properties of the derivation itself are part of the empirical content of the theory.

It is also useful to relate (1) to some of the existing work that has addressed similar issues. In a seminal paper Stabler (1997) presented a class of formalized minimalist grammars (MGs) and then wrote down a few derivations by hand that were argued to follow from them and thus “generate” some attested expressions. A more systematic approach along the same lines is presented in (Kobele, 2006). The derivational search function (1) automatizes this process so that the derivations are not done by hand, have no logical gaps (i.e., they do not rely on anything imprecise such as tree drawings), are error-free, executed extremely fast, and such that *everything* that follows from the grammar is in the output, not merely a few selected examples. In addition, because the derivational search function is implemented in Python we can test complex grammatical theories against nontrivial and large datasets in an instant.

Fong & Ginsburg (2019) and Ginsburg & Fong (2019) took a step further and implemented a “minimalist machine” which creates minimalist derivations from hierarchically organized lists of lexical items that are pre-ordered for converged computation and which make use of a decision procedure for selecting which operations are applied at which stages.[6] Thus, the system substitutes manually created derivations usually found from the literature with a more rigorous approach in which they are generated by a computational procedure. In the authors’ words, their work aims at “automatically deriving in full detail the examples that linguists typically use” (Fong & Ginsburg, 2019, p. 37). Moreover, the machine was tested in connection with several linguistic phenomena. The derivational search function (1) generalizes this idea by exploring the whole logical content of

[6] The lists are also organized hierarchically so that it becomes possible to derive embedded constituents such as DPs inside TPs: the DP is generated from its own sub-list and inserted inside the TP.

the theory: derivations are not selected in advance, lexical items are not pre-ordered, and there is no decision procedure (there can be derivational constraints, if envisioned by the linguist).[7] Furthermore, instead of verifying whether particular derivations work as intended, the idea is to match the grammar or hypothesis with nontrivial datasets collected by linguists interested in any particular phenomenon. The fact that there are particular derivations is an automatic consequence of the more general aim.

Let us illustrate (1) and its implementation with more concrete materials. In order to look at justification specifically we need some empirical claims to be justified. The choice is to some extent arbitrary. Here we examine the minimalist grammar originally designed by Chomsky (1995) and then developed by the author (Chomsky, 2000, 2001, 2008, 2013) and others cited later as we proceed. Let us assume, to make things as simple as possible, that the grammar contains only one operation, Merge, which combines syntactic objects X and Y and yields [X Y].[8] When applied to empty words *a*, *b*, *c* and *d* (=ad hoc lexical items that have no features), the derivational search function (1) generates 144 phrase structure representations (we also assume an additional left-to-right depth-first linearization algorithm which turns the output structures into linearized sentences). Here are the first eight (screenshots from the output of the script implementing the derivational search function):

[7] The authors state that a "free Merge" model in which the lexical items are not selected from a pre-ordered list must be non-deterministic (see p. 25). This might indeed be true if "free Merge" is defined as an indeterministic operation, but the derivational search function (1) which explores all possible ways of merging the material is and must be fully deterministic since the order in which various derivations are explored is well-defined.

[8] In some minimalist grammars phrase structures are binary sets {X Y} instead of asymmetric [X Y] as assumed in the main text (e.g., Collins & Stabler, 2016). The downside is that linearization as well as subcategorization/selection (and thus labeling/head algorithm) become nontrivial problems that we do not know at present how to solve and which would take us too far from the main topic. The source code repository contains a variation of the main script (**template1.py**) which uses the set-theoretical phrase structure formalism but does not have the linearization or labeling functions.

```
(1) c a b d  [ [ c [ a b]] d]
(2) d c a b  [ d [ c [ a b]]]
(3) a b c d  [ [ a b] [ c d]]
(4) c d a b  [ [ c d] [ a b]]
(5) a b c d  [ [ [ a b] c] d]
(6) d a b c  [ d [ [ a b] c]]
(7) c a b d  [ c [ [ a b] d]]
(8) a b d c  [ [ [ a b] d] c]
```

Consider the first two solutions. The derivational search function arrived at the first solution by generating [*c* [*a b*]] and merging *d* to the right = [[*c* [*a b*]] *d*]. Right before this operation was executed, the sWM contained two items {[*c* [*a b*]], *d*}. The algorithm then considers all operations of the grammar and applies them in a well-defined order to the two objects in sWM. Here there are two possible operations: (i) Merge([*c* [*a b*]] + *d*) and (ii) Merge(*d* + [*c* [*a b*]]). As shown by the above output list, it tried (i) first, followed by (ii). Since there were no more item to be merged, the algorithm backtracks and considered a derivation in which *d* is merged with *c* to generate [[*a b*][*c d*]] = (3). In this way, all possible derivations are explored systematically and exhaustively. Furthermore, when the script derives the sentences it writes all calculation steps into a separate logging file **log.txt** (stored permanently into **log_A.txt** that we use here). The first three steps of the derivation (written down by the algorithm) are shown below (all steps are on lines 1-2034):

```
 8  1.
 9
10      a, b, c, d
11
12      Merge(a, b)
13      = c, [_aP a b], d
14
15  2.
16
17      c, [_aP a b], d
18
19      Merge(c, [_aP a b])
20      = [_cP c [_aP a b]], d
21
22  3.
23
24      [_cP c [_aP a b]], d
25
26      Merge([_cP c [_aP a b]], d)
27      = [_dP [_cP c [_aP a b]] d]
28
29      [_dP [_cP c [_aP a b]] d]
30      ^ ACCEPTED: c a b d
```

We consult the derivational log file to find out what steps were taken during any derivation. In some cases we provide the line numbers for the log file used in this study.

The asymmetric binary-branching Merge is only one option among many, and there is no a priori reason why the human language faculty would rely on an operation of this type. What justifies it? The justification will be provided by the derivational search function (1) itself. Specifically, for now we assume that the asymmetric binary-branching bare phrase structure Merge as a tentative hypothesis – a speculative idea with no justification – and then justify it later by showing that correct empirical properties follow from it by using (1). In fact, this will be the sole *purpose* of (1). In addition, there will be no other justification: human intuition, phrase structure sketches or storyboarding or logically incomplete derivations will not be allowed to play a role. Thus, under the current methodology it is important to postpone the natural urge to find intuitive or incomplete justifications for theoretical assumptions until we have a useful and nontrivial datasets at hand that we can use to evaluate them. How exactly to do this is discussed below.

When discussing fully implemented theories it is useful and sometimes necessary to distinguish *computational*, *algorithmic* and *implementation level* descriptions from each other. The distinction was originally proposed by Marr (1982).

A computational level description is concerned with abstract mappings and ignores both the algorithm and the physical implementation. If we ignore the algorithm (1), what's left is an abstract mapping between a lexical selection and a set of pairs of linearized surface sentences and phrase structure analyses exhausting the logical content of the theory. The above system mapped {*a*, *b*, *c*, *d*} into 144 linearized sentence + phrase structure pairs shown above. This is what the theory (here, a trivial theory) claims matches with the empirical reality; it generates these pairings that we can match with reality. The same mapping could be generated by an unbounded number of different algorithms. We could generate the exact same mapping by beginning from the surface sentences, or consider a variation of (1) that mimics real language processing and/or comprehension. Another alternative would be a top-down derivation utilizing context-free rewriting rules. All approaches can implement the same computational level mapping. A useful rule of thumb is that computational level descriptions are evaluated against linguistic competence, algorithms against performance, but to test computational level descriptions rigorously we need *some* algorithm. Finally, the algorithm must be implemented so that it can be executed efficiently and in reasonable time by a physical system. Everything described in this article was implemented in Python. The implementation is again only one among many and as such introduces additional properties and details (e.g., language syntax, data-structures, class definitions) that are irrelevant to linguistics or to the argument developed in this article. Eventually we want to create a computational implementation theory for both the algorithm and the computational level description that does not contradict anything known independently about the human brain.

Testing the derivational search function with empty words is useful in making sure that the algorithm explores the whole derivational space, but unrealistic as a linguistic model. Real words have features such as subcategorization which restrict their distribution (Chomsky, 1965). For example, transitive verbs select DP-complements and not finite verbs. Subcategorization, in turn, relies on a labeling. Let us begin with the following (function **head** in the script):

(2) *Label/head for any syntactic object α*

a. If α is primitive, it will be the head;

b. suppose α = [X Y], then if X is primitive, it will be the head; otherwise

c. if Y is primitive, it will be the head; otherwise

d. the head of α is the head of Y (recurse into Y).

The head algorithm was added to the script, which changes the output into

```
(1) c a b d  [_dP [_cP c [_aP a b]] d]
(2) d c a b  [_dP d [_cP c [_aP a b]]]
(3) c d a b  [_aP [_cP c d] [_aP a b]]
(4) a b c d  [_cP [_aP a b] [_cP c d]]
(5) a b c d  [_dP [_cP [_aP a b] c] d]
(6) d a b c  [_dP d [_cP [_aP a b] c]]
(7) c a b d  [_cP c [_dP [_aP a b] d]]
(8) a b d c  [_cP [_dP [_aP a b] d] c]
```

where the calculated label L of complex syntactic object is denoted as [_LP X Y]. Once we know the head, we can add subcategorization. Let us stipulate two lexical features [+COMP:F] and [–COMP:F] which mandate and ban, respectively, feature F to/from the head of the complement of the lexical item having either feature.[9] This grammar will be able to regulate head-complement structures (e.g., *from the city*, **from bark, the dog, *dog the*), but cannot rule out ungrammatical specifier-head constructions, so we introduce [+SPEC:F] and [–SPEC:F] for specifier selection with the notion of specifier denoting all left phrases inside the projection from head X as determined by the above head algorithm (we discuss adjuncts in Section 5). It does not matter if both features are needed or whether this is the optimal way of defining them: as stated several times, all theoretical claims will be tested in a separate testing protocol discussed below. Moreover, the selected feature [F] can be syntactic or semantic; all we have to assume is that it is visible during the derivation.

Having introduced subcategorization, we must decide where it applies. The possibilities are: (i) at the stage where grammatical operations apply; (ii) at some intermediate construction ("phase", in the sense of (Chomsky, 2000, 2001)); (iii) at the final output. The first option reduces complexity; (iii) is also an obvious choice and will later morph into the interface legibility conditions. Assumption (ii) could be examined if the derivations were broken down into separate sub-derivations, but this option, though worth testing, was not implemented in this study. Assumption (i) requires that we block derivations violating subcategorization already inside the derivational search function. We added the test to (1), Step c.2.1, such that Merge(X, Y) is performed only if Y is compatible with the selection features of X. More generally the derivational search function was

[9] This system is intuitively clear but not optimal. We ignore this issue here since it will not play a role in the main argument.

implemented in such a way that each grammatical rule can be paired with its own preconditions that must be satisfied before the operation applies.

These assumptions allow us to test simple VP-grammars which enumerate sentences like *the dog bites the man* and *the man bites the dog* with the internal structure [$_{VP}$ [$_{DP}$ *the man*][$_{VP}$ *bites* [$_{DP}$ *the dog*]]] and rule out **the man the dog bites* or **the the dog bites man* and other word salads. Provided with reasonable subcategorization features (not discussed here since they can be found from the script itself) the grammar executes 128 derivational steps and generates the following 12 output sentences.

```
(1) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
(2) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(3) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(4) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(5) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
(6) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
(7) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(8) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(9) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
(10) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
(11) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(12) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
```

Note that the list has several identical sentences and phrase structures. This is because two separate lexical items were used for the definite article *the* (i.e. *the*$_1$ *dog…the*$_2$ *man* ≠ *the*$_2$ *dog…the*$_1$ *man*) and because the output counts derivations, not the final phrase structures or linearized sentences (mappings from derivations into output phrase structures and the latter into linearized sentences are all many-to-one). We could easily prune the output by removing duplicates, but at this stage we work with the raw output to illustrate the operation of (1). The following screenshot illustrates how the derivational search function derived the first accepted solution:

```
2043  1.
2044
2045      the, man, bite, the, dog
2046
2047      Merge(the, man)
2048      = the, bite, dog, [_DP the man]
2049
2050  2.
2051
2052      the, bite, dog, [_DP the man]
2053
2054      Merge(the, dog)
2055      = bite, [_DP the dog], [_DP the man]
2056
2057  3.
2058
2059      bite, [_DP the dog], [_DP the man]
2060
2061      Merge(bite, [_DP the dog])
2062      = [_VP bite [_DP the dog]], [_DP the man]
2063
2064  4.
2065
2066      [_VP bite [_DP the dog]], [_DP the man]
2067
2068      Merge([_DP the man], [_VP bite [_DP the dog]])
2069      = [_VP [_DP the man] [_VP bite [_DP the dog]]]
2070
2071      [_VP [_DP the man] [_VP bite [_DP the dog]]]
2072      ^ ACCEPTED: the man bite the dog
```

This constitutes one complete derivation from the initial numeration into one complete solution. The algorithm will then backtrack and examines all other possible derivations (lines 2038-3429 in the log file). For example, in the next steps it separates the verb and the two arguments and tries the combination *bite* + *the man*

```
2074  5.
2075
2076      bite, [_DP the dog], [_DP the man]
2077
2078      Merge(bite, [_DP the man])
2079      = [_DP the dog], [_VP bite [_DP the man]]
2080
2081  6.
2082
2083      [_DP the dog], [_VP bite [_DP the man]]
2084
2085      Merge([_DP the dog], [_VP bite [_DP the man]])
2086      = [_VP [_DP the dog] [_VP bite [_DP the man]]]
2087
2088      [_VP [_DP the dog] [_VP bite [_DP the man]]]
2089      ^ ACCEPTED: the dog bite the man
```

which will lead into solution (2) *the dog bite the man.*

Since the derivational search function performs exhaustive search, sentences (1–12) are the *only* sentences this grammar enumerates from the given numeration: all other permutations and structures are implicitly judged ungrammatical. Thus, the grammar judges **the man the dog bite* ungrammatical, although this fact is not explicitly stated anywhere.[10] This is an important feature of enumerative grammars that define the set of grammatical or attested expressions by literally generating its members: anything not generated is judged ungrammatical. This presupposes that the derivational search function performs exhaustive search.

Strictly speaking we haven't yet constructed any real proof that the grammar is correct – we eyeballed the output and compared it with the intended output we "had in mind." This is exactly what we want to avoid. We therefore provide the derivational search function with an explicit set of target sentences as input that we want the grammar to enumerate. Both the numeration and the target sentences can be provided in an external input file. For example, we can write the following three lines into the external input file to run the simple VP-grammar we just constructed against the two target sentences:

```
# 2
Numeration=the,man,bite,the,dog
the man bite the dog
the dog bite the man
```

The first line declares the numeration, the two that follow are the target sentences we want the model to generate from these lexical items. To use input of this type we just have to add a function

[10] Thus, this is an *enumerative* generative grammar which generates a set of expressions; everything else logically derivable from the same lexical items but not derived by the grammar is automatically judged ungrammatical. A *recognition grammar*, in contrast, decides for any given input whether it is grammatical or ungrammatical. Enumerative and recognition grammars are equivalent in the sense that it is possible to construct one from the other (as long as they compute in finite time), but they are tested in a different way: enumerative grammars are evaluated against their ability to enumerate all grammatical expressions and only them, recognition grammars are tested against datasets which contain both grammatical and ungrammatical expressions. In this article we examine enumeration grammars. For some recent work with recognition grammars that uses the methodology proposed in the present article, see (Brattico, 2019, 2020, 2021a, 2021b, 2022, 2023a, 2023b, 2024; Brattico & Chesi, 2020). Script **template4.py** provides a simple starting point for developing a recognition grammar with a derivational search function.

into the script that is able to parse the input. Running the model now *shows* that we reached observationally adequacy:

```
(11) the dog bite the man  [_VP [_DP the dog] [_VP bite [_DP the man]]]
(12) the man bite the dog  [_VP [_DP the man] [_VP bite [_DP the dog]]]
Derivational steps: 176
Errors 0
```

The script compares the output produced by the grammar automatically with the target sentences provided by the user in the input file and reports that there were no errors. External files like this will become the *datasets* that we will use to justify grammatical hypotheses.[11]

The assumption that we compare the output from the derivational search function with numeration-sentence pairs is inessential. We could compare any output generated by the algorithm (phonological strings, semantic interpretations, whole derivations, binding dependencies) with the intended output provided in some format in the dataset file. For example, we could write a function that provides all referential expressions appearing in the input sentences with their predicted thematic roles and then compare the predicted output with the correct thematic roles provided in the dataset. We could do the same with binding dependencies, or with any attribute that is of particular interest in any particular study. The main point is that we compare the predicted output of the algorithm with the attested output from native speakers *mechanically* so that (i) the size and complexity of the data will never be an issue and (ii) the procedure is deductive and does not rely on eyeballing or other subjective criteria. The derivational search function implements both properties when provided with an explicit dataset and a mechanical algorithm for performing comparisons between the input and output data, as was done here.[12]

[11] In the context of a realistic research project the dataset will be more complex and involve several numerations paired with the target sentences so that the hypotheses can tested against large batches of data. We call this type of computational experiments *batch testing*.

[12] It has been proposed that instead of pairing numerations with output expressions we should verify whether a set of expressions follows from a grammar + lexicon. The derivational search function can do this if we feed it with all possible numerations, given a lexicon. The system would then attempt to derive sentences like *Dogs bark* from numerations like {*Nixon*, *resigned*, *in*, *August*} among many other possibilities until it finally discovers {*dogs*, *bark*}. The problem is that

3 Lexicon, complex words and head chains

The simple VP-grammar relied on an intuitive notion of "word" that formed the basis of both the derivations and the output sentences. Technically these words were provided in their own data structure (call it the *lexicon*) as items defined by a set of lexical features, including the subcategorization features and major lexical categories, and were transformed into primitive syntactic objects that formed the initial numeration.[13] This meant that complex words such as *bites* were represented as such in the lexicon; the third person singular suffix and the present tense were represented as lexical features, though these features had no role in the grammar with just one operation Merge. This lexicalist assumption is not empirically implausible and characterizes a whole range of lexicalist approaches to word formation (Chomsky, 1970; Jackendoff, 1975; Aronoff, 1976; Lapointe, 1980; Anderson, 1982, 1992; Jensen & Stong-Jensen, 1984; Grimshaw & Mester, 1985; Di Sciullo & Williams, 1987; Borer, 1991; Zwicky, 1992; Sells, 1995; Bresnan & Mchombo, 1995; Scalise & Guevara, 2005; Kiparsky, 2017). In such theories the lexical items that form the starting point of syntactic derivations are generated by separate word formation rules. We could then either ignore word formation and write the words directly into the lexicon, or add word formation into the derivational search function to generate all "possible words" before phrasal syntactic composition takes over. Notice that the derivational search function uses a list of grammatical operations that do not have to be syntactic; they can be lexical as well.

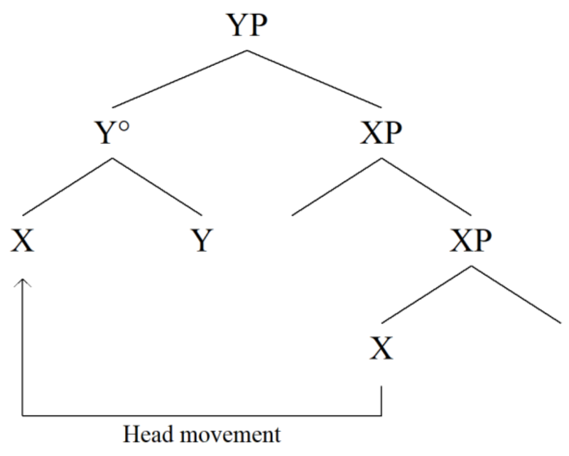
---

the method would be extremely slow and waste precious resources in exploring derivations that are blatantly irrelevant – for example, all numerations containing *Nixon* are irrelevant (unless we posit grammatical operations which delete or silence arbitrary content words). By providing the numeration in the dataset we restrict the amount of calculations to a plausible domain (notice that nothing prevents us from testing several numerations or by creating an automatic mechanism which considers some limited set of numerations).

[13] It does not matter if the lexical items are sets of lexical features or are represented by more complex structures. If the latter, then the lexical entries would be defined by a custom-made data structure that defines what these more complex entries are. It makes no difference to the overall argument.

Many generative theories also posit syntactic operations for the creation of complex words. Let us assume that there is an operation that creates tensed verbs by combining a syntactically less prominent verbal head with a tense head that selects the verb phrase as its complement. More generally, we assume that the operation produces $[_{YP}\ (_{Y}\ X\ Y)^0\ [_{XP} \ldots \cancel{X} \ldots]]$ where ~~X~~ is a silent copy of the head that has been copied and adjoined to the higher head Y (Koopman, 1984; Travis, 1984; Baker, 1985, 1988; Pollock, 1989; Borer, 1991; Hale & Keyser, 1993; Roberts, 2001; Julien, 2002: §2; Matushansky, 2006; Dékány, 2018). Schematically:

(3)



Head movement therefore represents a variation of Merge, but with two extra properties: it creates complex zero-level categories $(X\ Y)^0$ instead of regular phrases and "copies" something from an existing structure. Let's add this feature to the grammar to see what happens when it is combined with (1).

First we stipulate a feature 'zero-level' to distinguish complex zero-level categories $(_{Y}\ X\ Y)^0$ from complex phrasal categories $[_{\alpha P}\ X\ Y]$ and modify the head algorithm such that it responds to this property and not only to the lack of daughter constituents. That is, $(X\ Y)^0$ in the above will be a complex syntactic object with the status of a zero-level object.

Then we post an operation *Head Merge* which merges two constituents but creates a zero-level category instead of a regular phrase, the latter which is the output of regular Merge. Regular Merge creates [$_{\alpha P}$ X Y], Head Merge $(X\ Y)^0$. This operation will be part of the final head movement operation.

Next we determine how the properties of the newly created complex zero-level categories are calculated on the basis of their constituents. One possibility (the preferred one, in our view) is to change the head algorithm (2) so that it calculates heads also for complex zero-level categories and then let the features of complex heads depend on the features of their head, in the sense of Williams (1981), Selkirk (1982) and Di Sciullo & Williams (1987). Another solution is to posit separate feature inheritance inside the complex head. We used the latter as a heuristic solution since the former is nontrivial and requires a notion of adjunction that we will introduce later. Feature inheritance was implemented inside Head Merge and copies the features of Y to $(X\ Y)^0$.

Let us assume that the morpheme boundaries corresponding to Head Merge are represented by # in the output sentences.

If we just added Head Merge to the list of syntactic operations accessed by the derivational search function (1), new derivations will emerge in which the grammar produces new complex candidate words (e.g., *the#the*, *the#bite#man*) and inserts them into the derivations. Although this is one possible starting point when modeling word formation and not empirically impossible if the operation is controlled in some way,[14] this does not match with the original specification α = [$_{YP}$ ($_{Y}$ X Y)^0 [$_{XP}$ ...~~X~~...]] (3) which copies X from within the merge partner XP.

One possibility is that head movement is a grammatical operation like Merge and applies freely: (3) will become the composite operation of Merge(Y XP) + head movement (X…~~X~~) which *may* apply in a sequence but need not be (or even Merge + Copy + Head Merge, if also copying is modelled as an independent operation). This presupposes that the objects selected as targets for

[14] Several minimalist models assume an operation of this type (Embick, 2004, 2023; López, 2015; Wood, 2023; Brattico, 2024). We left the Head Merge into the algorithm as a free operation for future development, but controlled its application in such a way that it plays no role in the data discussed in this paper (no complex words like *the#the* were generated). We ignore this whole issue in this article in the interest of keeping the material succinct.

syntactic operations inside the derivational search function include not only the syntactic objects in the syntactic working memory but also the daughters of the said elements.[15] The approach maintains full generality and consequently opens up derivations where all syntactic operations can target the internal constituents of the syntactic objects in the syntactic working memory. It also introduces an enormous amount of derivational paths that we never see in reality, for example, the derivational search function will attempt to perform sideward head movement. Another consequence is that the operation is countercyclic: it assumes that X can be inserted inside [Y XP] which requires that we open up previously established phrase structure.[16] Finally, since we will later examine the complexity properties of realistic linguistic theories we should aim for grammars which minimize complexity. The above solution is extremely costly in terms of complexity.

A second solution is to posit unitary syntactic operations into the derivational search function which targets single objects and tampers with their internal properties. Head movement, for example, would target some X, locate its head (or all heads) and apply the operation either freely or as a reflex of some triggering condition. This hypothesis still requires countercyclic operations, but it generates less superfluous derivations than the first solution. The derivational search function was generalized in such a way that it can handle the second solution, should this hypotheses be developed and set up for rigorous testing.

There is a third alternative which does not increase complexity and seems to provide the correct empirical properties: make head movement part of Merge. We can assume that right before [Y XP] is created head movement applies to X^0 (assuming the Head Movement Constraint (Koopman, 1984; Travis, 1984)) if Y is a zero-level object with a feature making it a bound morpheme. The operation then copies X and merges it with Y by Head Merge introduced earlier, silences the

[15] Either all daughters or daughters selected by some additional criteria. Regardless of the choice, enormous amounts of new derivations will open up.

[16] Specifically, a countercyclic derivation must (i) detach Y from its mother β, (ii) create a new constituent $\alpha = (X\ Y)^0$ by Head Merge and (iii) insert α as the new daughter for β. Operations (i, iii) must be added to the formalism, while (ii) is Head Merge.

original X phonologically and creates [$_{YP}$ (X Y)^0 [$_{XP}$ ...~~X~~...] by Merge(Y, XP). Intuitively, the head is moved from one syntactic object into another "across the syntactic working memory" right before the objects become one. The independent steps are: (i) consider Y, XP as candidates for Merge; (ii) copy the head X from XP (if applicable) and merge it to Y to yield (X Y)^0; (iii) create [$_{YP}$ (X Y)^0 XP]. The operation is cyclic and applies under restricted contexts: sideward head copying are excluded and the operation satisfies HMC. The operation does not increase complexity, since it applies whenever Merge applies and does not involve derivational branching. Is this assumption justified empirically?

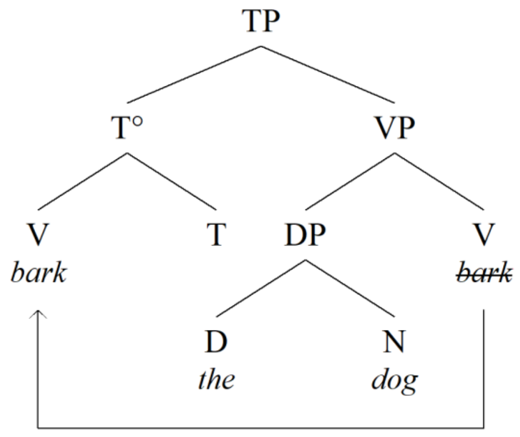
To test it, let us posit a bound tense head T* into the lexicon (we use the asterisk for experimental morphemes that are posited for testing purposes and later replaced with more realistic units) and assume that it selects a VP to create tensed verb phrases. To verify this grammar against data we assume the numeration {*T**, *the*, *dog*, *bark*} where *bark* is an intransitive verb, provide a target sentence *the dog bark#T** (item #3 in the dataset) and feed them to the derivational search function.[17] Running the grammar with these assumptions (lines 3433-3521 in the log file) produces the following result:

```
(1) bark#T* the dog  [_TP (bark T*) [_VP [_DP the dog] __:1]]
Derivational steps: 10
Errors 2
Should not generate: {'bark#T* the dog'}
Should generate: {'the dog bark#T*'}
```

The solution is illustrated further in (4).

[17] It is assumed that *bark#T** would be replaced with the vocabulary item *barks*, but this process, which is trivial to implement by brute force, is not useful here because it would mask the derivational history of the word. Moreover, in a more realistic grammar the spellout portion of the derivation could involve complex postsyntactic operations of its own (Marantz, 1984, 1997; Halle & Marantz, 1993; Embick & Noyer, 2001; Hale & Keyser, 2002; Matushansky, 2006; Embick & Marantz, 2008; Harizanov & Gribanova, 2018; Harizanov, 2018).

(4)



The experiment verifies that the grammar created an expected complex tensed verb (V, T)^0 by copying the verb from within the VP. The trace of the copying operation is marked by __:1. But as it did so, it generated a VS order that was not among the target sentences. The reason this grammar did not put the argument to the preverbal SpecTP position was because we assumed in the lexicon that intransitive verbs must have DP specifiers.[18] If we remove this assumption, the grammar generates the correct word order *the man barks* but with a questionable analysis (1–2) where the argument is merged directly to SpecTP:

```
(1) the dog bark#T*  [_TP [_DP the dog][_TP (bark T*) __ ]]
(2) the dog bark#T*  [_TP [_DP the dog][_TP (bark T*) __ ]]
(3) bark#T* the dog  [_TP (bark T*)[_VP [_DP the dog] __ ]]
Derivational steps: 10
Errors 1
Should not generate: {'bark#T* the dog'}
Should generate: set()
```

This grammar is observationally but not descriptively adequate. It produces correct output sentences but the underlying syntactic analyses (and derivations) are implausible on independent grounds. It is usually assumed that all thematic arguments must be merged inside the VP to receive a thematic role (Fukui & Speas, 1986; Sportiche, 1988; Koopman & Sportiche, 1991), and that the

[18] Intransitive verbs can be divided at least into two categories, those which requires a DP-specifier ("unergatives") and those which require a DP-complement ("unaccusatives"). The choice does not matter for the matter at hand.

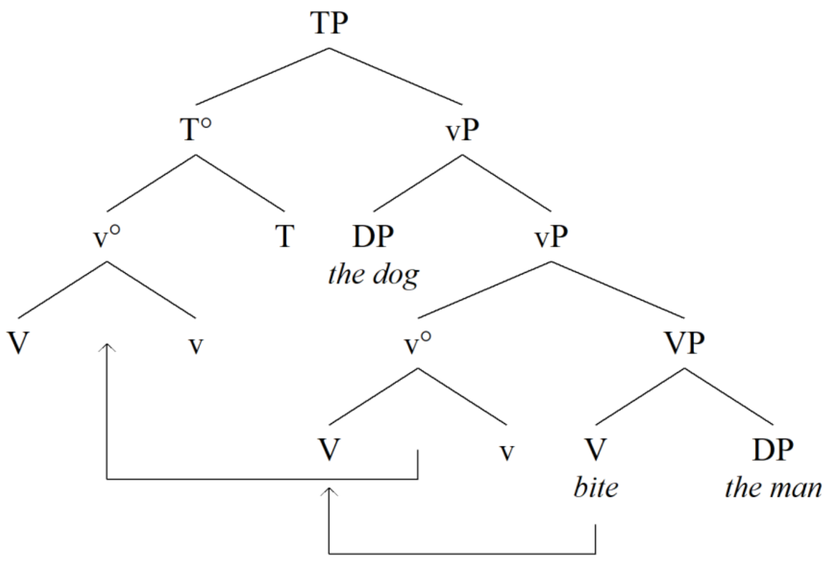
subcategorization features of verbal elements are checked inside the VP. The two first analyses do did satisfy this principle. We return to the problem of English V-initial sentences below.

Transitive verbs are usually analyzed as being created by merging verbal stems with a transitivizer v (or a separate Voice head, Kratzer (1996), see also Pylkkänen (2008)) which introduces the external (agent, causer) argument to its specifier, so that all thematic arguments are initially generated inside the vP where they receive thematic roles and satisfy subcategorization. Consequently we introduce a bound morpheme v into the lexicon such that it requires a DP-specifier and selects for a VP. Adding v into the numeration (item #4 in the dataset) generates 16 sentences after 2275 derivational steps (lines 3525-21342), of which the first four are as follows:

```
(1) bite#v#T* the dog the man  [_TP ((bite v) T*) [_vP [_DP the dog] [_vP __:1 [_VP __:2 [_DP the man]]]]]
(2) bite#v#T* the man the dog  [_TP ((bite v) T*) [_vP [_DP the man] [_vP __:1 [_VP __:2 [_DP the dog]]]]]
(3) bite#v#T* the dog the man  [_TP ((bite v) T*) [_vP [_DP the dog] [_vP __:1 [_VP __:2 [_DP the man]]]]]
(4) bite#v#T* the dog the man  [_TP ((bite v) T*) [_vP [_DP the dog] [_vP __:1 [_VP __:2 [_DP the man]]]]]
```

Notice the familiar “snowball” profile of the three zero-level categories making up the final tensed transitive verb (5), a consequence of the way head movement was defined.

(5)



The subcategorization features of v and V require that the two DPs are merged inside the vP, which again leaves the verb at the first position. Because head movement was defined as a local operation, this grammar cannot derive sentences in which heads skip over potential targets or where they combine with lower heads.

To show that the grammar is adequate in a linguistically interesting sense requires that we test it against many more numerations. For example, if we simulate this grammar with a numeration that lacks v (#5 in the dataset, lines 21346-27081), it will pass SVO and VSO sentences with analyses $[_{\text{TP}}$ T $[_{\text{VP}}$ DP $[_{\text{VP}}$ V DP]]] and $[_{\text{TP}}$ DP [T $[_{\text{VP}}$ DP V]] that one might and most likely will consider implausible. Thus, the derivational search function found that the grammar is leaking, but this became evident only after we tested it with additional numerations. Because the derivational search function exhausts the logical implications of the theory, it will eventually find all leaks provided it is tested with adequate and representative materials.

English does not generate V-initial sentences, but some other languages do. To model crosslinguistic variation we need a theory of the said variation and some technical implementation for the notion of "different grammar." The easiest way to accomplish a system of this type is to create several language-specific speaker models instantiating the system and then use them on the basis of the language in the numeration and/or input sentence. Each speaker model would contain a different lexicon (depending on the language) plus different grammar rules and/or parameters, depending on how the variation is modeled. If the variation is modelled by positing different rules (implausible, but possible), then the list of grammatical rules for each speaker model should be populated from a larger pool of possible rules when the speaker model instances are generated; if the rules differ only in terms of finite number of parameters, then each speaker model would contain a data-structure holding the values of these parameters in addition to the rules. If languages differ only in terms of their lexicon, then the solution is to pair each speaker model with a different lexicon. These assumptions were implemented in the script underlying this article. See Appendix A.

The point of the above exercise was not to argue for any particular model of head movement or word formation, but rather to show how to add grammatical operations to the theory and let the derivational search function to apply them to capture concrete datasets to determine what the new operations (e.g., head movement) imply in conjunction with existing operations (Merge). Reversing the process, first we create a representative and systematic dataset that we think bears on the questions of interest and then, with that dataset at hand, examine how various theoretical alternatives play out. The iterative process of hypothesis formation and testing will constitute a real justification for some analysis or hypothesis – not a priori reasoning or intuitive sketching.

4 Phrasal movement

The final experiment performed in Section 3 produced verb-initial sentences that are ungrammatical in English, though not in all languages. The traditional solution going back to (Chomsky, 1981, 1982) is to assume that functional heads can have a special nonthematic specifier selection feature, call it EPP, which forces them to have a specifier that is not merged directly to the specific position but is copied from within the complement to create α = [$_{YP}$ ZP [$_{YP}$ Y [$_{XP}$ ~~ZP~~ [X WP]]]]. Implementing an operation of this type without countercyclic derivations requires that we instantiate the operation after Merge has created [Y XP], where Y has the EPP-feature.[19] Adding the rule directly to Merge suffices to eliminate VS(O) sentences without increasing complexity. For example, the numeration {*T*, *the*, *dog*, *bark*} (#6) will produce the following output in which the DP-argument is dislocated to SpecTP due to the lexically stipulated EPP-feature at T (lines 27085-27173):

```
(1) the dog bark#T  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]]
Derivational steps: 10
Errors 0
```

[19] The two other solutions introduced in the previous section in connection with head movement are possible as well and would seem to merit full examination. We could add phrasal movement to the catalog of grammatical operations applied at every derivational step and then allow all operations to target both the syntactic objects in the working memory and their internal constituents. Or we could create a special category of rules which affect the internal structure of single syntactic objects. Perhaps the claim that the operations apply "freely" (Chomsky, 2008) refers to an architecture of this type.

The part of the derivation generated by the derivational search function and leading into the accepted output is as follows.

```
27132  6.
27133
27134      T, the, dog, bark
27135
27136      Merge(the, dog)
27137      = [_DP the dog], T, bark
27138
27139  7.
27140
27141      [_DP the dog], T, bark
27142
27143      Merge([_DP the dog], bark)
27144      = T, [_VP [_DP the dog] bark]
27145
27146  8.
27147
27148      T, [_VP [_DP the dog] bark]
27149
27150      Merge(T, [_VP [_DP the dog] bark])
27151           + Head chain by T° targeting bark°
27152           + Phrasal A chain by (bark T)° targeting [_DP the dog]
27153      = [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]]
27154
27155      [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]]
27156      ^ ACCEPTED: the dog bark#T
```

T is combined with the verb to create a finite tensed verb (line 27151); the resulting complex verb triggers an A-movement and copies the grammatical subject to SpecTP (line 27152). This operation is caused by the EPP feature at T as specified in the lexicon. A transitive variant is item #7 in the dataset and replicates the process. The verb-initial solution is removed from the output, so the grammar is at least observationally adequate. Is the solution also descriptively adequate? It is hard to say: EPP-induced A-movement has generated substantial debate (Fernández-Soriano, 1999; Chomsky, 2000, 2001, 2008; Holmberg, 2000; Miyagawa, 2001, 2010; Bošković, 2002, 2007; Rezac, 2004; Epstein & Seely, 2005; Rackowski & Richards, 2005; Landau, 2007) which makes evaluation difficult. It should be noted that these mechanisms do not yet correlate A-movement with any "anchoring" properties (Landau, 2007) such as topicness, agreement or case assignment, and we did not restrict the type of phrases that can be targeted: the fact that the grammatical subject was moved was an accidental consequence of the fact that it was merged to SpecvP and thereby formed the required local [T vP] configuration for A-movement. This is an obvious problem, but not an implausible starting point since there are languages, such as Finnish (Holmberg & Nikanne, 2002), in which the EPP feature can be checked by nonsubject topics. Again, to move forward we should create a representative dataset that bears on the nature of A-movement and only test various alternatives by using the derivational search function.

The problem is even more challenging, however, since there are phrasal movement constructions that differs fundamentally from the subject movement illustrated above. One is operator movement, observed in English interrogatives such as *which man*$_1$ *did the dog bite* __$_1$. Moreover, English interrogativization is associated with Aux-inversion, which is also visible in yes/no questions (*did the dog* __ *bite the man?*). Thus, the simple A-movement illustrated above is clearly not sufficient.

Let us first fix the issue with the auxiliary and then consider the issue of Ā-movement. Suppose *did* represents T and that the force of the sentence (e.g., interrogative, declarative, imperative) is represented by an abstract C head subcategorizing for TP. We can test the results with dataset (#9)

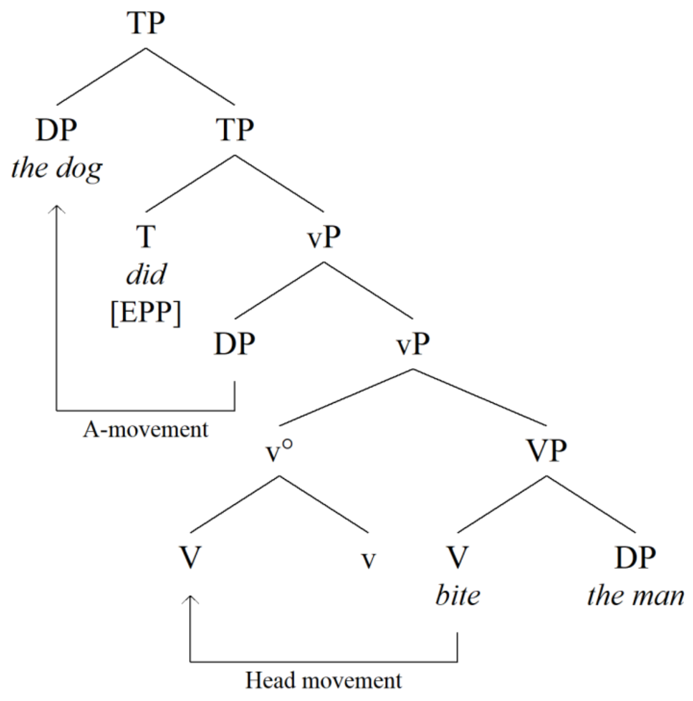
```
Numeration=C,the,man,did,v,bite,the,dog
C the man did bite#v the dog
C the dog did bite#v the man
```

which derives

```
(1) C the dog did bite#v the man  [_CP C [_TP [_DP the dog]:3 [_TP did [_vP __:3 [_vP (bite v) [_VP __:2 [_DP the man]]]]]]]
(2) C the man did bite#v the dog  [_CP C [_TP [_DP the man]:6 [_TP did [_vP __:6 [_vP (bite v) [_VP __:5 [_DP the dog]]]]]]]
(3) C the dog did bite#v the man  [_CP C [_TP [_DP the dog]:19 [_TP did [_vP __:19 [_vP (bite v) [_VP __:18 [_DP the man]]]]]]]
(4) C the dog did bite#v the man  [_CP C [_TP [_DP the dog]:3 [_TP did [_vP __:3 [_vP (bite v) [_VP __:2 [_DP the man]]]]]]]
```

and 12 other sentences, all generating one of the target sentences (lines 61789-270480). See (5) for illustration.

(6)

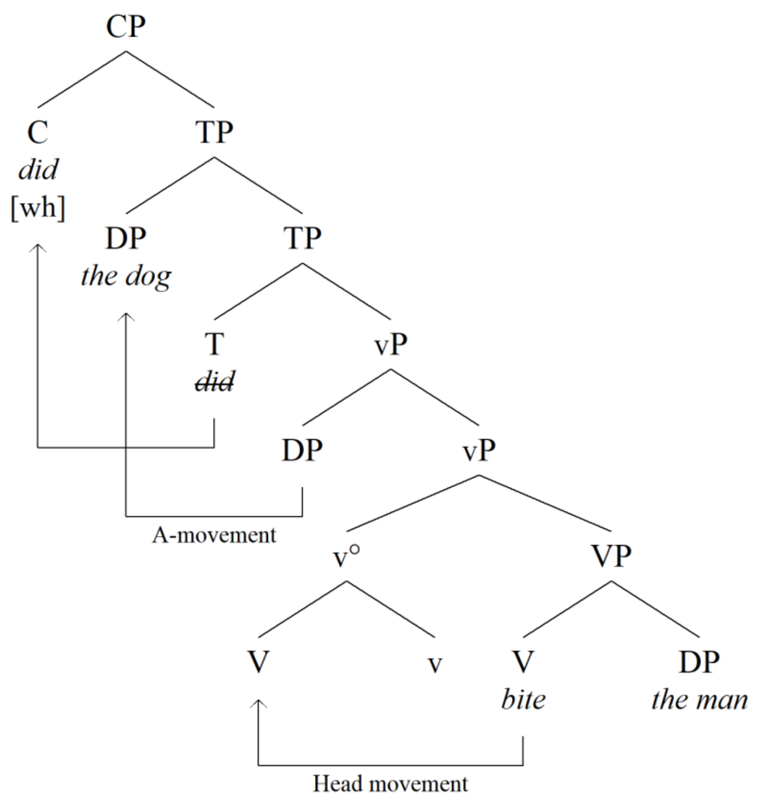


Since tense is now expressed by *did*, the verb remains at a lower position. All we have to do to model English Aux-inversion under current assumptions is to assume that an interrogative C is a bound morpheme, which generates the Aux-inversion pattern (item #10 in the dataset, lines 270484-415549)[20]:

```
(1) did#C(wh) the dog bite#v the man  [_CP (did C(wh)) [_TP [_DP the dog]:1 [_TP __:2 [_vP __:1 [_vP (bite v) [_VP __:7 [_DP the man]]]]]]]
(2) did#C(wh) the dog bite#v the man  [_CP (did C(wh)) [_TP [_DP the dog]:1 [_TP __:2 [_vP __:1 [_vP (bite v) [_VP __:4 [_DP the man]]]]]]]
(3) did#C(wh) the dog bite#v the man  [_CP (did C(wh)) [_TP [_DP the dog]:1 [_TP __:2 [_vP __:1 [_vP (bite v) [_VP __:5 [_DP the man]]]]]]]
(4) did#C(wh) the man bite#v the dog  [_CP (did C(wh)) [_TP [_DP the man]:2 [_TP __:3 [_vP __:2 [_vP (bite v) [_VP __:7 [_DP the dog]]]]]]]
```

[20] If we remove the auxiliary the model produces verb-inversion (*bites#v#T#C the dog the man?*) which is ungrammatical in English but grammatical in many languages, so the derivation is not completely implausible, though something must block it in English.

(7)



Having these mechanisms in the model we can model interrogativization. The numeration will be {*C(wh)*, *the*, *man*, *did*, *v*, *bite*, *which*, *dog*} and the target sentences are as follows (items #11–12):

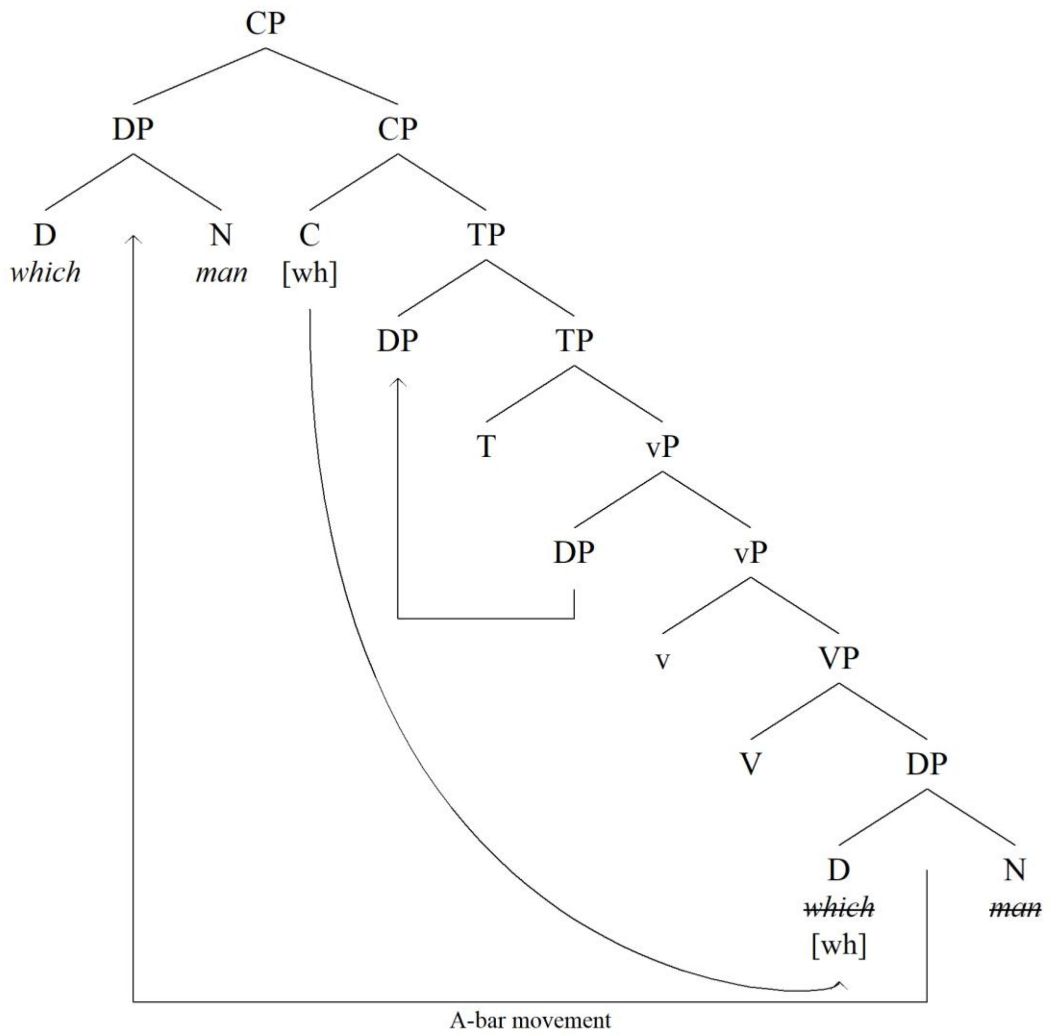
```
Numeration=C(wh),the,man,did,v,bite,which,dog
which dog did#C(wh) the man bite#v
which man did#C(wh) the dog bite#v
which dog did#C(wh) bite#v the man
which man did#C(wh) bite#v the dog
```

Let us assume that C(wh) has an interrogative wh-feature [wh] which triggers search for the corresponding interrogative operator that we assume to be D with the same feature (=*which*). Let us assume, furthermore, that after Merge has created [X YP], X is checked for the interrogative feature and copy and merge is attempted. The search algorithm, call it *minimal search* after Chomsky (2008), moves downstream following the projectional spine of each head H and then enters the complement of H if any, returning first constituent with a wh-feature at its head.[21] This grammar enumerates the following outputs (among others):

```
(1) which dog did#C(wh) bite#v the man  [_CP [_DP which dog]:2 [_CP (did C(wh)) [_TP __:2 [_TP __:3 [_vP __:2 [_vP (bite v) [_VP __:6 [_DP the man]]]]]]]
(2) which dog did#C(wh) the man bite#v  [_CP [_DP which dog]:4 [_CP (did C(wh)) [_TP [_DP the man]:2 [_TP __:3 [_vP __:2 [_vP (bite v) [_VP __:5 __:4]]]]]]]
(3) which dog did#C(wh) bite#v the man  [_CP [_DP which dog]:2 [_CP (did C(wh)) [_TP __:2 [_TP __:3 [_vP __:2 [_vP (bite v) [_VP __:6 [_DP the man]]]]]]]
(4) which dog did#C(wh) bite#v the man  [_CP [_DP which dog]:1 [_CP (did C(wh)) [_TP __:1 [_TP __:2 [_vP __:1 [_vP (bite v) [_VP __:6 [_DP the man]]]]]]]
```

which is the correct, intended result. Both the external argument and the internal argument can be moved to SpecCP, depending on where the interrogative operator *which* is. (8) illustrates the operations (probing and movement):

(8)



[21] See the function **minimal search** in the source code. Informally the search continues to X in {X X Y} if X is complex, to Y if it is selected by X.

Notice that if the interrogative pronoun was the subject, it was first moved to SpecTP and only then to SpecCP in a process called successive-cyclic movement (i.e. A-movement followed by A-bar movement). This consequence is unavoidable if phrasal movement operates cyclically in connection with Merge.

Because minimal search follows labeling and head-complement configurations, we suspect that the CED-effects (Huang, 1982) follow. However, the numeration posited above does not quite show this, since the derivational search function positioned all operators on the minimal search paths due to independent requires. We must craft a numeration from a sentence such as [*the dog* [*from which city*] *barks*] and show that **which city*$_1$ [*the dog* [*from* __$_1$]] *barks* is not in the output. First, adding the preposition *from* into the grammar with reasonable subcategorization and testing it with an intransitive numeration together with C and T (#13) produced 84 accepted derivations (lines 560340-976993), among them the irrelevant solutions (9)a–b and the relevant analyses (9)c–d.

(9) a. C the dog barks from the city. (Cf. 'the police came from the city.')

b. From the city C the dog barks. (PP base-generated to SpecCP)

c. C the city from the dog barks. (Implausible interpretation)

d. C the dog from the city barks.

Let us ignore the irrelevant results; once we know that the relevant analysis (9)d is in the output we can replace one definite article with *which* and the declarative C with an interrogative C(wh) and run the model to verify that **which city the dog from* __ *barks* is not among the output (item #14 in the dataset, lines 976997-1166053). The derivational search function executes 23613 calculation steps, which shows that the grammar does not allow extraction from subjects. However, the experiment also generated a few ungrammatical sentences such as those in (10) which show that the model is not observationally adequate.

(10) a. [$_{PP}$ from the dog [$_{CP}$ which city$_1$ C(wh) __$_1$ barks]] (cf. 'from the dog which barks')

b. From which city C(wh) barks the dog? (base-generated PP to SpecCP)

c. [$_{CP}$ From the dog [$_{CP}$ which city C __ barks]]? (Double-filled SpecCP)

d. Barks$_1$ the dog __$_1$ from which city? (cf. 'Does the dog bark from which city?')

We will again ignore these problems, since they are not relevant for the issue at hand except that they show how the derivational search function will automatically relevel all problems in our grammar or hypothesis.

It is also worth noting in passing that these assumptions generate successive-cyclic A-movement provided that there is a sequence of local heads with the EPP feature. For example, the numeration {*the*, *dog*, *T*, *seem*, *to*, *bark*} (item #15, lines 1166058-1170261) generates the following output if the raising verb and the infinitival *to* have the EPP feature:

```
(1) the dog seem#T to bark  [_TP [_DP the dog]:3 [_TP (seem T) [_VP __:3 [_VP __:1 [_T/infP __:3 [_T/infP to [_VP __:3 bark]]]]]]
Derivational steps: 523
Errors 0
```

This property allows us to create a kernel for English personal passives. If we assume a special passive v* with an unthematic specifier position (EPP) instead of an external thematic agent argument, we get sentences in which the direct object is moved successive-cyclically to the SpecTP position (#16, lines 1170266-1170567):

```
(1) the man was bite#v*  [_TP [_DP the man]:2 [_TP was[_VP __:2 [_VP (bite v*)[_VP __ __:2 ]]]]]
Derivational steps: 58
Errors 0
```

The numeration contains *was* = T and the special passive v*, (bite v*)^0 would be spelled out as *bitten* (see note 17). The direct object is raised successive-cyclically from the VP into Specv*P and then to SpecTP.[22]

[22] The standard minimalist theory posits a third operation, Agree, responsible for feature covariance dependencies such as subject-verb agreement. The above system already contains a kernel of an operation of this type: we assumed that C(wh) searches for a matching operator element inside its complement before phrasal movement is executed. This was implemented by a minimal search function which takes a (wh-)feature as input and locates a head that has the same

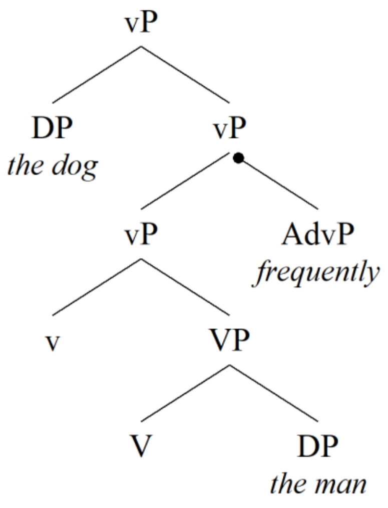
5 Adjunction

In this section we formalize adjuncts as “parallel objects” lingering in the syntactic working memory until linearization (see (Hunter, 2015) for a similar theory of adjuncts).

Consider a sentence such as *the dog bites the man frequently*. This construction behaves as a regular transitive clause but with an additional modifier *frequently* that is invisible for all head calculations within the projectional spine of the transitive clause, optional and occurs in a rightward position that cannot be the complement position of the verb, being filled by the patient. Semantically the adverb modifies the whole event, not the man, the dog or biting; what happens frequently is that the dog bites the man. Another property is that we can add more adverbials to the clause, in principle without restrictions thought there are semantic and pragmatic limitations. If we assume that bare events are represented by VPs, then the adverb should be connected in some way to the verb phrase, but there is nothing in our grammar that connects anything directly with phrasal level representations. Overall there is nothing in the model that reproduces any of these properties.

The syntactic working memory contains a set of “parallel” syntactic objects that we can think of being under active consideration at any given movement during the derivation. Let us assume that there is an operation Adjoin(X, Y) which connects X to Y via one-way mother-of dependency so that Y will be the mother of X but the two elements are otherwise left intact in the sWM. Thus, the mother of *frequently* could be a verb phrase while all other properties of the verb phrase, including its daughter constituents, remain the same (adjuncts are marked by •—):

feature. The operation resembles long-distance Agree (or probe-goal dependency, in some more recent models). It is possible that a theory of Agree could be developed from these mechanisms without positing a separate operation. The approach would rule out upward-directed agreement systems (Chomsky, 1993; Koopman, 2006; Chandra, 2007; Baker, 2008; Merchant, 2011; Zeijlstra, 2012; Carstens, 2016; Bjorkman & Zeijlstra, 2019; Baker & Camargo Souza, 2020; Keine & Dash, 2022) and is therefore neither empirically inconsequential nor something that could be assumed without giving it careful consideration.



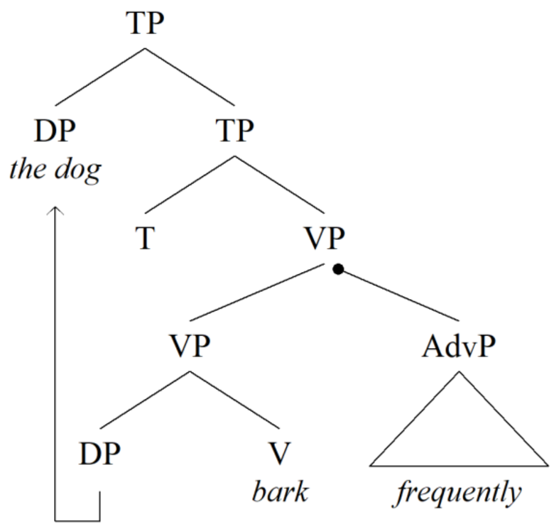
Both the vP and AdvP remain in sWM after Adjoin(AdvP, vP). Not all syntactic objects can be adjoined, and those which can be adjoined can and often do have a limited distribution. For example, we want to create a theory in which VP-adverbs such as *frequently* must be adjoined inside the VP but not inside PP. To restrict the class of adjoinable objects and their distributions let us assume that adjoinability is defined by a feature [α:F] where F must occur inside the head of the projection to which the adjunct is linked with.[23] A VP-adverb can be created by [α:V]. Only syntactic objects headed by this feature can be adjoined. We do not assume the vice versa: adjoinable objects can also be merged, but must then satisfy subcategorization.[24] Let us consider a simple intransitive sentence *the dog barks frequently* with the numeration {T, *the*, *dog*, *bark*, *frequently*} and where we assume that the adverbial has [α:V] (item #17, lines 1170572-1171254). Given all of the above the derivational search function produces the following:

```
(1) the dog bark#T frequently  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 [_VP __:1 frequently]]]]
(2) frequently the dog bark#T  [_AdvP frequently [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]]]
(3) the dog bark#T frequently  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]] + { frequently }
(4) the dog bark#T frequently  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]] + { frequently }
(5) the dog bark#T frequently  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 [_VP __:1 frequently]]]]
(6) the dog bark#T frequently  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]] + { frequently }
```

[23] A possible generalization is α:F_1, . . ., F_n in which several features must be checked.
[24] This property can easily be removed from the grammar, but is arguable needed to model properties of DPs and PPs which can assume both roles.

Parallel adjuncts are represented as + {X, Y, ...} after the root structure. Solutions (1, 5) are generated by merging the adverbial to the complement position of the verb. Solution (2) is generated by merging the adverb above TP, but although this sentence is grammatical the analysis is not plausible since the head algorithm will regard the whole phrase as an adverb phrase.[25] All label/head calculations inside the host structure remain the same because the adjunct is invisible. Solutions (3, 4, 6) involve adjunction into VP, structures such as X.

(11)



Running the model with a transitive numeration (item #19, lines 1172496-1423980, 32278 steps) shows that the adverb is always positioned between the verb and the object (**the dog bite frequently the man*). In the above simulations we assumed that the adjunct is always linearized to the left of its host. It appeared at the post-verbal position because we nowhere assumed that verb copying should update adjunct links; the adjunct is linked to the silenced verb at the base position and is linearized in relation to that element as shown in (12).

(12) The dog (bark$_1$, T) [$_{\text{VP}}$ frequently __$_1$].

[25] This consequence can be avoided if we assume that *frequently* is derived as [$_{\text{AdvP}}$ (*frequent-ly*)^0[$_{\text{AP}}$ ~~*frequent*~~]], but we do not go into the details here since the issue is irrelevant for the main point.

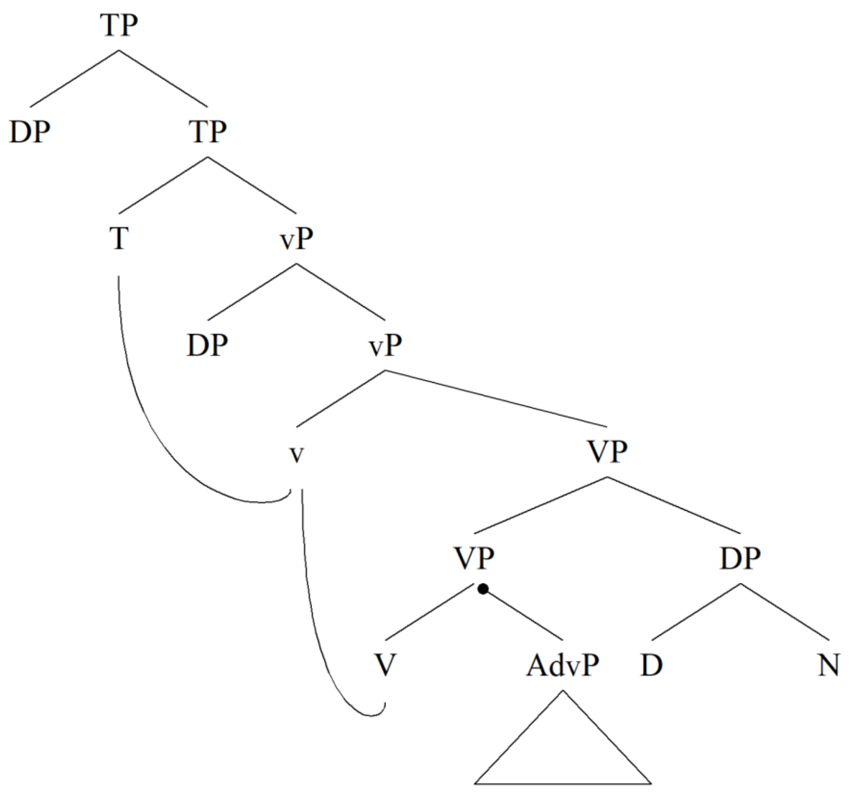
Changing linearization to right will create problems later if we want to have leftward adjuncts such as adjectives. Since linearization of adjuncts is subject to language-specific variation, we can perhaps assume that it is controlled by a lexical linearization feature. If we assume that adverbs like *frequently* are linearized to the right, the grammar will generate the variants in (13)(item #19).

(13) a. The dog bites the man frequently. (Adjoined to VP)

b. Frequently the dog bites the man. (Merged to SpecTP)

However, it will also generate postverbal adverbial sentences such as *The dog bites frequently the man* when the adverbial is right-adjoined to V (14). That is, it is a feature of the current grammar that adjunction to V positions the adverb between the verb and the direct object, adjunction to VP after the direct object.

(14)



If adjectives are linearized to left, they will occupy a position between D and N. A numeration {*the*, *angry*, *dog*, *bark*, T} (item #21) will generate the following output if we restrict adjective distribution to NPs:

```
(1) the angry dog bark#T  [_TP [_DP the dog]:2 [_TP (bark T) [_VP __:2 __:1]]] + { angry }
Derivational steps: 57
Errors 0
```

The derivation for *the angry dog* is as follows:

```
 4.

     the, angry, dog

     Adjoin(angry, dog)
     = dog, the + { angry }

 5.

     dog, the + { angry }

     Merge(the, dog)
     = [_DP the dog] + { angry }

     [_DP the dog] + { angry }
     ^ ACCEPTED: the angry dog
```

First *angry* is adjoined to *dog* (step 4), then the definition article is merged with N to generate [$_{\text{DP}}$ D N] with the adjective existing as a parallel object linked with N (step 5). Linearization to left creates the final output sentence. We could create Italian style postnominal adjectives by assuming that they linearize to the right of their host.

Having adjunction in the model opens up several new derivations, for example, we can grant PPs an ability to right-adjoin to verb phrases. A vP-numeration such as {*the*, *dog*, *v*, *bite*, *the*, *man*, *in*, *the*, *city*} (item #21b) generates 13608 verb phrase configurations after 4,5 million calculation steps if we allow the PP to right-adjoin to any verb; an intransitive variation is much easier to calculate (item #22, 1424501-1441608). It produces the expected output, part of which is shown below:

```
(15) the dog bark in the city  [_VP [_DP the dog] bark] + { [_PP in [_DP the city]] }
(16) the dog bark in the city  [_VP [_DP the dog] [_VP bark [_PP in [_DP the city]]]]
(17) the dog in the city bark  [_VP [_DP the [_NP dog [_PP in [_DP the city]]]] bark]
```

Linearization was set to right for all English prepositions; leftward PPs are still possible via base-generation to the specifier position whenever the operation is licensed by subcategorization. Replacing *the* = *which* demonstrates that the other half of the CED-effects also follows: there are no derivations in which the interrogative is extracted from an adjoined PP, only subjects and objects can be fronted to SpecCP (item #23, lines 1441612-2356498, after 116049 calculation steps). However, the model does not handle pied-piping and therefore strands the preposition (15).

(15) ?*Which city$_1$ did the dog bark [$_{PP}$ in __$_1$]] (see e.g., line 1743102).

Notice that the correct sentence *in which city did the dog bark?* is also in the output set, but it is generated by base-generating the interrogative PP into the SpecCP and could be considered implausible.[26]

6 Discussion and summary

We have constructed a rudimentary computational grammar with several key components such as Merge, subcategorization, labeling, asymmetric binary-branching bare phrase structure, A-movement, Ā-movement, head movement, head merge and adjunction. Although there was nothing new in these components as such, indeed they were extracted from the literature essentially without modification, we have constructed a computational infrastructure consisting of a derivational search function + grammar + lexicon that can be used to test grammatical hypotheses. Let us imagine, then, that we attempt to construct the proofs *without* any automatization, by some paper-and-pencil methodology. How many calculation steps would be required? Table 1 summarizes the amount of calculations required to verify the test grammar as a function of the size of the numeration or data complexity (items #1-28 in the dataset). We provide the numbers both for realistic words and empty words (ad hoc words with no features), the latter which represents a particularly hostile complexity

[26] How to capture pied-piping and prevent stranding is a separate problem. There is a large literature addressing this matter that we cannot attempt to review here.

profile.[27] Notice that both the head and phrasal movement were implemented as part of Merge and did not increase complexity. Moreover, the numbers provided for the realistic simulation concern the particular grammar formulated in this article. This table therefore provides a reasonable estimate of what to expect when working with realistic grammars.

Table 1. Number of calculation steps as a function of the size of the numeration and data complexity.

		Number of steps	
Size	Example	Realistic	Empty words
1	*the*	0	0
2	*the man*	1	2
3	*the dog barks*	2	18
	the angry dog	5	
4	*the dog barks#T*	10	228
	the angry dog barks	10	
	the dog barks frequently	26	
5	*C the dog barks#T*	96	4580
	the angry dog barks#T	57	
	the dog barks#T frequently	83	
	the angry dog barks frequently	153	
	the man bite the dog	176	
	C(wh) which dog did bark?	62	
	*the man was bitten#v**	38	

[27] Not the worst case scenario, since generalizing head and phrasal movement could potentially create even larger search spaces.

6	*C the angry dog barks#T*	680	137430
	the angry dog barks#T frequently	668	
	T the man bite the dog	721	
	the dog seem#T to bark	523	
	the dog barks in the city	2152	
7	*the dog bite#T#v the man*	2275	5772102
	the angry dog bite#v the man	11171	
	the dog did bite#v the man	2275	
8	*C the dog bite#T#v the man*	27280	hundreds of millions
	the dog bite#T#v the man frequently	32278	
	C did the dog bite#v#T the man?	18872	
	C(wh) did the man bite#v the dog?	18690	
	C the dog from the city barks#T	53511	
	C the dog from which city barks#T	23613	
	C which dog barks#T in the city	116049	
9	*C the angry dog bite#T#v the man*	246510	billions
10	*the dog bite#v#T the man in the city*	650749	

These numbers suggest that the number of calculations increases at least by the order of *!n* as a function of the size of the numeration *n*. Testing the grammar with a slightly more complex but still completely realistic numeration such as *the man believes that the dog barks* (#30) generates 5 million calculations steps, 40 million lines of logging. The whole log file containing the calculations and results reported in this article has 9 million lines, which approximates the amount of "notebook pages" required to perform the inferences manually. Notice that we are still talking about relatively simple numerations and constructions shown in Table 1. Furthermore, to produce the research reported in this article all these calculations were performed several hundred times, perhaps thousands. All paper-and-pencil methods therefore quickly become infeasible.[28] This leaves us with two options: either use less precision (i.e. fall back on the prescientific intuitive method) or to use computation. The latter is obviously what we should do: the derivational search function is trivial to write for any grammar that is sufficiently rigorous to allow full formalization, we can easily use it to test grammars against any amount of data, and the results are replicable, transparent and are not contaminated by subjective factors such as theoretical preferences, background assumptions or confirmation biases. In fact, the method reaches the minimal standards of rigor (16) assumed in all advanced sciences.

(16) *Methodological principle*

Theories, data and justifications must be unambiguous.

We therefore propose (16) as a methodological principle. We have shown that it is feasible, simple to use, and trivial to implement.

[28] The Python script calculated at the rate of one million steps/two minutes when we removed all printing and logging. We used the standard Python (3x) libraries. The calculations can be performed much faster with optimized Python libraries or by relying on lower-level languages such as C. Physical calculation speed will eventually become meaningless, however, unless the search tree is also restricted logically. Notice that the complexity profiles documented here do not depend on the fact that the theory was formalized, or on the fact that the theory was implemented in Python. The issue cannot be eliminated by working with informal sketches. The complexity profile stems from the combinatorial nature of the theory and data. As shown in this article, the problem persists even if we add empirically motivated properties to the empirical theory.

APPENDIX A

Python implementation of the derivational search function

The model elucidated in the main article was implemented in Python (3x), a general-purpose programming language well-suited for this type of problems due to its simplicity and relatively high-level of abstraction. The code is written inside one short script that was designed in such a way as to provide in this authors' view an optimal starting point for more realistic and ambitious linguistic projects. See Appendix B for the explanation of the script(s) and replication.

The script is organized around four classes: phrase structure class (**PhraseStructure**) which defines the basic properties of the underlying phrase-structure formalism; speaker model class (**SpeakerModel**) which generates a model of a speaker/hearer and which executes the grammatical tasks such as derivational search; language data class (**LanguageData**) which maintains and perform data processing, including file handling, that is needed independent of any linguistic subject matter, and lexical class (**Lexicon**) which handles all lexical processing. In addition, the raw lexicon is defined by its own dictionary structure and lexical redundancy rules at the beginning of the script. The idea behind this division is that the speaker model class would be used to model different speakers (languages, dialects) while the phrase structure class defines the universal properties of phrase structure formalism.

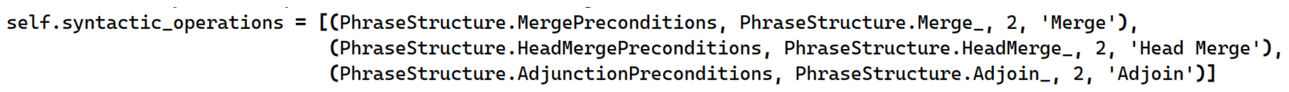
The script works by reading the dataset (a list of numerations and target sentences, as described in the main text), creating a speaker model to implement the derivational search and then by calling **run_study** which sends all numerations to the speaker model for processing:

```
ld = LanguageData()              #   Instantiate the language data object
ld.read_dataset('dataset3.txt')  #   Name of the dataset file processed by the script, reads the file
sm = SpeakerModel()              #   Create default speaker model, would be language-specific in a more realistic model
run_study(ld, sm)                #   Runs the study
```

The derivational search function, shown below, is inside the speaker model class and is initially called with the numeration.

```
def derivational_search_function(self, sWM):
    if self.derivation_is_complete(sWM):
        self.process_final_output(sWM)
    else:
        for Preconditions, OP, n, name in self.syntactic_operations:
            for SO in itertools.permutations(sWM, n):
                if Preconditions(*SO):
                    PhraseStructure.logging_report += f'\n\t{name}({self.print_lst(SO)})'
                    new_sWM = {x for x in sWM if x not in set(SO)} | tset(OP(*tcopy(SO)))
                    self.consume_resource(new_sWM, sWM)
                    self.derivational_search_function(new_sWM)
```

The input parameter sWM contains a set of syntactic phrase structure objects (as defined by the phrase structure class). Before the derivation begins, the lexical words in the numeration must be mapped into syntactic objects by using the lexicon. This is done by a separate **lexical retrieval** function, which maps lexical items (from the lexical data structure) into primitive syntactic objects. These objects are then fed to the derivational search function when the derivation begins. The higher level **if-else** structure in the derivational search function defines the halting condition: derivation is halted when no more processing can be done. What this means will depend on the particular assumption of the grammar, so it is defined inside its own function. The function **process_final_output** handles all post-syntactic processing and is responsible for checking the output for syntactic/semantic well-formedness, linearization plus printing ("spelling out") and storing the results. If there were semantic interpretation, it would be part of this function. In terms of the standard minimalist theory, the derivation would split here into two pathways, one going towards the PF-interface, another for LF-interface. The derivational search function implements recursive branching. It considers all syntactic operations OP_n of the grammar (outer **for**-loop) and all ways of creating *n* syntactic objects from the contents of the sWM (inner **for**-loop), applies $OP_n(X_1, . . ., X_n)$, updates sWM, and calls the function recursively. The syntactic operations available in the grammar are stored as a list in **syntactic_operations**. For example, in the current system we have three operations, Merge, Head Merge and Adjoin:



```
self.syntactic_operations = [(PhraseStructure.MergePreconditions, PhraseStructure.Merge_, 2, 'Merge'),
                             (PhraseStructure.HeadMergePreconditions, PhraseStructure.HeadMerge_, 2, 'Head Merge'),
                             (PhraseStructure.AdjunctionPreconditions, PhraseStructure.Adjoin_, 2, 'Adjoin')]
```

The rules implementing head movement and phrasal movement are not listed here since we assumed that they are part of Merge; if we wanted them to apply freely as independent rules, they would appear in this list. Each rule is a tuple that contains a function defining its preconditions ("triggering conditions"), the operation itself, number of arguments the operation takes and a name for the operation that will appear in the log file. Notice that the logic of the derivational search function is that every operation is applied at every step, thus the preconditions do not license the operations but block unwanted application. Syntactic objects are selected from the sWM by the function **permutations** that is part of the **itertools** module imported in the beginning of the script. As seen in the code, the number of syntactic objects selected will depend on the number of arguments required by the syntactic rule itself. The function returns a tuple SO of syntactic objects. If the selected objects satisfy the preconditions for the rule, the rule is applied and the sWM is updated before the recursive call. An important detail is that before OP is applied to $X_1, . . ., X_n$ the syntactic objects are copied: we want each recursive branching to work with independent objects.

APPENDIX B
The python scripts

The project **Template** (https://github.com/pajubrat/Templates) contains several template scripts that have been written as convenient and simple starting points for adding the derivational search function into linguistic projects. The script used in the study reported in this article is **template2.py**. Running this script replicates the output that was discussed in this article. It creates a separate **log.txt** file. The derivations underlying the results reported in the present article are stored permanently into **log_A.txt** and will not get overwritten. Script **template1.py** contains a starting point for a set-theoretical Merge that was not developed further in this work. **template4.py** provides a derivational search function and a starting point for a recognition grammar, not discussed in the present article. The scripts named as **template2.x** are discussed elsewhere, but in essence they develop the enumerative grammar further and improve it in many other ways.

Anderson, S. R. (1982). Where's Morphology? *Linguistic Inquiry*, *4*, 571–612.

Anderson, S. R. (1992). *A-morphous morphology*. Cambridge University Press.

Aronoff, M. (1976). *Word formation in generative grammar*. Cambridge, MA.: MIT Press.

Baker, M. (1985). The Mirror Principle and Morphosyntactic Explanation. *Linguistic Inquiry*, *16*, 373–415.

Baker, M. (1988). *Incorporation. A theory of grammatical function changing*. University of Chicago Press.

Baker, M. (2008). The Syntax of Agreement and Concord. In *The Syntax of Agreement and Concord*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511619830

Baker, M., & Camargo Souza, L. (2020). Agree without Agreement: Switch-reference and reflexive voice in two Panoan languages. *Natural Language and Linguistic Theory*, *38*(4), 1053–1114. https://doi.org/10.1007/S11049-019-09463-W/METRICS

Bjorkman, B. M., & Zeijlstra, H. (2019). Checking Up on (ϕ-)Agree. *Linguistic Inquiry*, *50*(3), 527–569. https://doi.org/10.1162/ling_a_00319

Borer, H. (1991). The causative-inchoative alternation: A case study in parallel morphology. *Linguistic Review*, *8*, 119–1158.

Bošković, Ž. (2002). A–movement and the EPP. *Syntax*, *5*(3), 167–218. https://doi.org/10.1111/1467-9612.00051

Bošković, Ž. (2007). On the Locality and Motivation of Move and Agree: An Even More Minimal Theory. *Linguistic Inquiry*, *38*, 589–644.

Brattico, P. (2019). *Computational implementation of a linear phase parser. Framework and technical documentation (version 15.0)*. Pavia.

Brattico, P. (2020). Finnish word order: does comprehension matter? *Nordic Journal of Linguistics*, *44*(1), 38–70. https://doi.org/doi:10.1017/S0332586520000098

Brattico, P. (2021a). A dual pathway analysis of information structure. *Lingua*, *103156*.

Brattico, P. (2021b). Null arguments and the inverse problem. *Glossa: A Journal of General Linguistics*, *6*(1), 1–29. https://doi.org/10.5334/gjgl.1189

Brattico, P. (2022). Predicate clefting and long head movement in Finnish. *Linguistic Inquiry*, *54*(4), 663–692. https://doi.org/http://dx.doi.org/10.1162/ling_a_00431

Brattico, P. (2023a). Computational analysis of Finnish nonfinite clauses. *Nordic Journal of Linguistics*, 1–40. https://doi.org/10.1017/S0332586523000082

Brattico, P. (2023b). Structural case assignment, thematic roles and information structure. *Studia Linguistica*, *77*(1), 172–217. https://doi.org/10.1111/stul.12206

Brattico, P. (2024). External head Merge and derivational word formation. *Manuscript*.

Brattico, P., & Chesi, C. (2020). A top-down, parser-friendly approach to operator movement and pied-piping. *Lingua*, *233*, 102760. https://doi.org/10.1016/j.lingua.2019.102760

Bresnan, J., & Mchombo, S. A. (1995). The lexical integrity principle: Evidence from Bantu. *Natural Language and Linguistic Theory*, *13*(2), 181–254. https://doi.org/10.1007/BF00992782

Carstens, V. (2016). Delayed Valuation: A Reanalysis of Goal Features, “Upward” Complementizer Agreement, and the Mechanics of Case. *Syntax*, *19*(1), 1–42. https://doi.org/10.1111/SYNT.12116

Chandra, P. (2007). *Dis(Agree): Movement and agreement reconsidered* [Ph.D. thesis]. University of Maryland.

Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA.: MIT Press.

Chomsky, N. (1970). Remarks on Nominalization. In R. A. Jacobs & P. S. Rosenbaum (Eds.), *Readings in English Trasformational Grammar*. Wiley.

Chomsky, N. (1981). *Lectures in Government and Binding: The Pisa Lectures*. Dordrecht: Foris.

Chomsky, N. (1982). *Some concepts and consequences of the theory of Government and Binding*. MIT Press.

Chomsky, N. (1993). A minimalist program for linguistic theory. In K. Hale & S. J. Keiser (Eds.), *The view from building 20*. Cambridge, MA.: MIT Press.

Chomsky, N. (1995). *The Minimalist Program*. Cambridge, MA.: MIT Press.

Chomsky, N. (2000). Minimalist Inquiries: The Framework. In R. Martin, D. Michaels, & J. Uriagereka (Eds.), *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik* (pp. 89–156). Cambridge, MA.: MIT Press.

Chomsky, N. (2001). Derivation by Phase. In M. Kenstowicz (Ed.), *Ken Hale: A Life in Language* (pp. 1–37). Cambridge, MA.: MIT Press.

Chomsky, N. (2008). On Phases. In C. Otero, R. Freidin, & M.-L. Zubizarreta (Eds.), *Foundational Issues in Linguistic Theory: Essays in Honor of Jean-Roger Vergnaud* (pp. 133–166). Cambridge, MA.: MIT Press.

Chomsky, N. (2013). Problems of projection. *Lingua*, *130*, 33–49.

Collins, C., & Stabler, E. P. (2016). A Formalization of Minimalist Syntax. *Syntax*, *19*(1), 43–78.

Dékány, É. (2018). Approaches to head movement: A critical assessment. *Glossa: A Journal of General Linguistics*, *3*(1). https://doi.org/10.5334/gjgl.316

Di Sciullo, A. M., & Williams, E. (1987). *On the definition of the Word*. Cambridge, MA.: MIT Press.

Embick, D. (2004). On the structure of resultative participles in English. *Linguistic Inquiry*, *35*, 355–392.

Embick, D. (2023). Smaller syntax for English stative passives: A first report. *Acta Linguistica Academica*, *70*, 285–316.

Embick, D., & Marantz, A. (2008). Architecture and Blocking. *Linguistic Inquiry*, *39*(1), 1–53. https://doi.org/10.1162/ling.2008.39.1.1

Embick, D., & Noyer, R. (2001). Movement Operations after Syntax. *Linguistic Inquiry*, *32*(4), 555–595. https://doi.org/10.1162/002438901753373005

Epstein, S., & Seely, D. (2005). *Transformations and Derivations*. Cambridge: Cambridge University Press.

Fernández-Soriano, O. (1999). Two Types of Impersonal Sentences in Spanish: Locative and Dative Subjects. *Syntax*, *2*(2), 101–140. https://doi.org/10.1111/1467-9612.00017

Fong, S., & Ginsburg, J. (2019). Towards a Minimalist Machine. In R. C. Berwick & E. P. Stabler (Eds.), *Minimalist Parsing* (pp. 16–38). Oxford: Oxford University Press.

Fukui, N., & Speas, M. (1986). Specifiers and projection. *MIT Working Papers in Linguistics*, *8*, 128–172.

Ginsburg, J., & Fong, S. (2019). Combining linguistic theories in a Minimalist Machine. In R. C. Berwick & E. P. Stabler (Eds.), *Minimalist Parsing* (pp. 39–68). Oxford: Oxford University Press.

Graf, T. (2013). *Local and transderivational constraints in syntax and semantics* [PhD thesis]. UCLA.

Grimshaw, J., & Mester, R.-A. (1985). Complex verb formation in Eskimo. *Natural Language & Linguistic Theory*, *3*(1), 1–19.

Hale, K., & Keyser, S. J. (1993). On Argument Structure and the Lexical Expression of Syntactic Relations. In K. Hale & S. J. Keyser (Eds.), *The View from Building 20: Essays in Linguistics in Honor of Sylvain Bromberger* (pp. 53–109). MIT Press.

Hale, K., & Keyser, S. J. (2002). *Prolegomenon to a theory of argument structure*. Cambridge, MA.: MIT Press.

Halle, M., & Marantz, A. (1993). Disrtibuted Morphology and the pieces of inflection. In K. Hale & S. J. Keyser (Eds.), *The view from Building 20: Essays in Honor of Sylvain Bromberger* (pp. 11–76). MIT Press.

Harizanov, B. (2018). Word Formation at the Syntax-Morphology Interface: Denominal Adjectives in Bulgarian. *Linguistic Inquiry*, *49*(2), 283–333. https://doi.org/10.1162/LING_a_00274

Harizanov, B., & Gribanova, V. (2018). Whither head movement? *Natural Language & Linguistic Theory*, *37*(2), 461–522.

Holmberg, A. (2000). Scandinavian Stylistic Fronting: How any Category can become an Expletive. *Linguistic Inquiry.*, *31*(3), 445–484. https://doi.org/10.1162/002438900554406

Holmberg, A., & Nikanne, U. (2002). Expletives, subjects and topics in Finnish. In P. Svenonius (Ed.), *Subjects, Expletives, and the EPP* (pp. 71–106). Oxford: Oxford University Press.

Huang, C.-T. J. (1982). Move wh in a language without wh movement. *Linguistic Review*, *1*, 369–416.

Hunter, T. (2015). Deconstructing Merge and Move to Make Room for Adjunction. *Syntax*, *18*(3), 266–319. https://doi.org/10.1111/synt.12033

Jackendoff, R. (1975). Morphological and Semantic Regularities in the Lexicon. *Language*, *51*(3), 639. https://doi.org/10.2307/412891

Jensen, J. T., & Stong-Jensen, M. (1984). Morphology is in the lexicon! *Linguistic Inquiry*, *15*(3), 474–498.

Julien, M. (2002). *Syntactic Heads and Word Formation*. Oxford: Oxford University Press.

Keine, S., & Dash, B. (2022). Movement and cyclic Agree. *Natural Language and Linguistic Theory*, 1–54. https://doi.org/10.1007/S11049-022-09538-1/METRICS

Kiparsky, P. (2017). Nominal verbs and transitive nouns: Vindicating lexicalism. In C. Bowern, L. Horn, & R. Zanuttini (Eds.), *On looking into words (and beyond)* (pp. 311–345). Berlin: Language Science Press.

Kobele, G. M. (2006). *Generating Copies: An investigation into Structural Identity in Language and Grammar* [PhD thesis]. UCLA.

Koopman, H. (1984). *The Syntax of Verbs: from Verb Movement Rules in the Kru Languages to Universal Grammar*. Foris.

Koopman, H. (2006). *Agreement configurations: In defense of the "Spec head"* (pp. 159–200). John Benjamins.

Koopman, H., & Sportiche, D. (1991). The position of subjects. *Lingua*, *85*(2/3), 211–258.

Kratzer, A. (1996). *Severing the External Argument from its Verb*. 109–137. https://doi.org/10.1007/978-94-015-8617-7_5

Landau, I. (2007). EPP Extensions. *Linguistic Inquiry, 38*, 485–523.

Lapointe, S. (1980). *A theory of grammatical agreement.* [Ph.D. dissertation]. University of Massachusetts-Amherst.

López, L. (2015). Parallel Computation in Word Formation. *Linguistic Inquiry, 46*(3), 657–701.

Marantz, A. (1984). *On the Nature of Grammatical Relations*. MIT Press.

Marantz, A. (1997). No escape from syntax: Don't try morphological analysis in the pricacy of your own lexicon. In A. Dimitriadis, L. Siegel, C. Surek-Clark, & A. Williams (Eds.), *Proceedings of the 21st Annual Penn Linguistics Colloquium* (pp. 201–225). Philadelphia: University of Pennsylvania, Penn Linguistics Club.

Matushansky, O. (2006). Head Movement in Linguistic Theory. *Linguistic Inquiry, 37*, 69–109.

Merchant, J. (2011). Aleut case matters. In *Pragmatics and autolexical grammar: In honor of Jerry Sadock* (pp. 193–210). Amsterdam: John Benjamins. https://doi.org/10.1075/LA.176.12MER

Miyagawa, S. (2001). Scrambling, EPP, and wh-in situ. In M. Kenstowicz (Ed.), *Ken Hale: A Life in Language* (pp. 293–338). MIT Press.

Miyagawa, S. (2010). *Why Agree? Why Move? Unifying Agreement-Based and Discourse-Configurational Languages*. Cambridge, MA.: MIT Press.

Pollock, J. Y. (1989). Verb Movement, Universal Grammar and the structure of IP. *Linguistic Inquiry, 20,* 365–424.

Pylkkänen, L. (2008). *Introducing arguments*. Cambridge, MA.: MIT Press.

Rackowski, A., & Richards, N. (2005). Phase Edge and Extraction: A Tagalog Case Study. *Linguistic Inquiry, 36*(4), 565–599. https://doi.org/10.1162/002438905774464368

Rezac, M. (2004). The EPP in Breton: An unvalued categorial feature. In H. van Riemsdijk, H. van der Hulst, & J. Koster (Eds.), *Triggers* (pp. 451–492). Berlin: Mouton de Gruyter.

Roberts, I. (2001). Head movement. In M. Baltin & C. Collins (Eds.), *Handbook of syntactic theory* (pp. 113–147). Oxford: Blackwell.

Salvati, S. (2011). Minimalist grammars in the light of logic. In S. Pogodalla, M. Quatrini, & C. Retoré (Eds.), *Logic and grammar* (pp. 81–117). New York: Springer.

Scalise, S., & Guevara, E. (2005). The lexicalist approach to word-formation and the notion of the lexicon. In P. Stekauer & R. Lieber (Eds.), *Handbook of word-formation* (pp. 147–187). Dordrecht: Springer.

Selkirk, E. (1982). *The Syntax of Words*. Cambridge, MA.: MIT Press.

Sells, P. (1995). Korean and Japanese Morphology from a Lexical Perspective. *Linguistic Inquiry*, *26*(2), 277–325.

Sportiche, D. (1988). A Theory of Floating Quantifiers and its Corollaries for Constituent Structure. *Linguistic Inquiry*, *19*(3), 425–449.

Stabler, E. P. (1997). Derivational minimalism. In C. Retoré (Ed.), *Logical Aspects of Computational Linguistics (LACL 96)* (pp. 68–95). Berlin Heidelberg: Springer.

Stabler, E. P. (2011). *Computational Perspectives on Minimalism* (C. Boeckx, Ed.). Oxford: Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199549368.013.0027

Travis, L. (1984). *Parameters and Effects of Word Order Variation* [Ph.D. thesis]. MIT.

Williams, E. (1981). On the Notions “Lexically Related” and “Head of a Word.” *Linguistic Inquiry*, *12*, 245–274.

Wood, J. (2023). *Icelandic Nominalizations and Allosemy*. Oxford: Oxford University PressOxford. https://doi.org/10.1093/oso/9780198865155.001.0001

Zeijlstra, H. (2012). There is only one way to agree. *Linguistic Review*, *29*(3), 491–539. https://doi.org/10.1515/TLR-2012-0017

Zwicky, A. M. (1992). Some choices in the theory of morphology. In *Formal Grammar: Theory and Implementation* (Levine, R., pp. 327–371). Vancouver: Oxford University Press.