Why movement comes for free once you have adjunction

Thomas Graf
Stony Brook University

1 Introduction

1.1 Overview

This paper presents a novel answer to the question why Move might be an integral part of language. The answer is rooted in the computational framework of subregular complexity, which has already been fruitfully applied to phonology. The computational perspective reveals that Merge belongs to the class TSL (*tier-based strictly local*) if the grammar also allows for recursive adjunction. Any cognitive device that can handle this level of computational complexity also possesses all the resources that are needed for Move. In fact, Merge and Move are remarkably similar when viewed as instances of TSL. Consequently, Move has no additional computational or conceptual cost attached to it and comes essentially for free.

1.2 Background

Every version of Minimalist syntax distinguishes at least two operations: Merge, and Move. Merge represents the ability to build larger structures from smaller ones and thus is indispensable for language. Move, on the other hand, captures the displacement property, i.e. that sometimes parts of a sentence are pronounced in a position that is different from the position where they are interpreted. In order to explain why Move is also a fundamental property of languages, Chomsky (2004) reduces Move to Merge by defining it as the process of merging a structure S with a proper subpart P of S. Movement of P is thus reanalyzed as re-merger of P. As a result, Move is no more complex than Merge and comes for free in any system that has the computational resources to carry out Merge operations.

Recent computational findings on the *subregular complexity* of Merge and Move call the validity of this reduction into question. Subregular complexity is concerned with measuring the power of computational devices that are weaker than the already fairly simple finite-state machines. Surprisingly, it seems that such devices are still expressive enough for natural language (Heinz 2009; Heinz & Idsardi 2013; Chandlee 2014; Jardine 2016; Aksënova *et al.* 2016; Graf 2017; a.o.). While subregular complexity has mostly been defined for string-based systems, it can be extended to trees and thus to syntax. Results that have done so have found a pronounced difference between Merge and Move.

Graf (2012) and Graf & Heinz (2015) have shown that in the formal framework of Minimalist grammars (MGs; Stabler 1997, 2011), Merge belongs to the class *strictly local* (SL), whereas Move falls at least into the more powerful class TSL, which enhances SL with tree tiers. The complexity of Move can even go beyond TSL depending on how one formalizes intermediate movement. That Merge is SL and Move TSL suggests that the reduction in Chomsky (2004) proceeds in the

wrong direction: Move is not a special case of Merge, but Merge is a computationally limited, special case of Move.

This paper provides a solution to this challenge by reevaluating the subregular complexity of Merge. While it is true that Merge is SL, this holds only for grammars without adjunction. In grammars with adjunction, there is no longer an upper bound on the distance between a head and its arguments. As a result, Merge moves beyond SL into the class TSL, which is also occupied by Move. It follows that Merge and Move are of comparable complexity. Remarkably, though, it also holds that their respective computations are almost exactly the same. Therefore, Move can indeed be regarded as a simple and very natural generalization of Merge.

1.3 Structure of the paper

The paper works its way towards this conclusion as follows: I first discuss the result of Graf (2012) that Merge is in SL (§2). The result itself is less important than the method by which it is obtained: the structure-building operation Merge is converted into a conjunction of constraints on MG derivation trees, and each constraint is in turn equated with a set of well-formed derivation trees in order to measure its subregular complexity. The same method is used in §3 to show that even though Merge is still SL in MGs with limited versions of adjunction (§3.1, §3.2), recursive adjunction pushes the subregular complexity of Merge from SL to TSL (§3.3, §3.4). In §4, I then contrast the TSL-view of Merge against the TSL-view of Move as developed in Graf & Heinz (2015). Based on this comparison, I conclude that the two operations are almost exactly the same, mirroring Chomsky's reduction of Move to Merge at a computational level.

2 Complexity of Merge

Let us first consider the complexity of Merge in isolation, following in the footsteps of Graf (2012). This requires a computational model of Merge, which is provided by MGs. MGs are closely modeled after Minimalist syntax, although they differ in some respects. I will point out such differences whenever they are crucial for the results of this paper. This will be rare, though, as the core insights apply to any variant of Minimalism that adopts some version of subcategorization and feature-driven movement.

I first discuss how Merge works in MGs in general (§2.1) and how it can be reanalyzed as constraints on derivation trees (§2.2). I then show that Merge (in MGs without adjunction) is a locally bounded dependency and thus belongs to the very simple class SL (§2.3). These results provide the backdrop for the subsequent discussion of Merge in MGs with adjunction (§3).

2.1 Merge in Minimalist grammars

Following MG tradition, but contrary to some recent proposals in Minimalist syntax, we assume that Merge does not apply freely but is mediated by a feature-driven subcategorization mechanism. That is to say, every LI has a category feature X^{-} and possibly one or more selector features Y^{+} that encode what arguments the

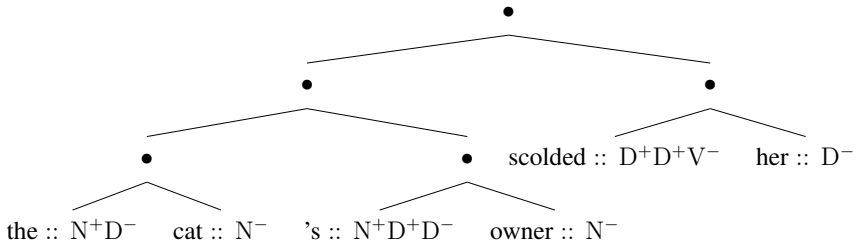
LI requires. These category and selector features fully control the application of Merge.

For example, the noun *cat* has only one feature, the category feature N^-. Therefore it can be selected by any LI that is looking for a noun, but it cannot take any arguments of its own. The determiner *a*, on the other hand, has the selector feature N^+ and the category feature D^-. More precisely, *a* carries the feature string N^+ D^- — the order of the features indicates that the determiner first has to merge with a noun before it can merge with an LI that is looking for a DP. The same logic dictates that the feature specification of the possessive marker is N^+ D^+ D^- as it first merges with the possessee NP, then with the possessor DP, and only then can it act as a DP and merge with an appropriate selector.

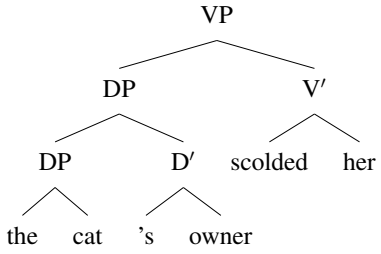
While feature ordering is never explicitly assumed in Minimalist syntax, there is an implicit consensus that whatever mediates selection exercises tight control over the order of arguments. This is why, say, v selects a VP as its complement and a subject DP as its specifier, rather than the other way around. The ordered Merge features of MGs thus are closely in line with linguistic practice, despite initial appearance to the contrary. For each LI, its string of selector and category features describes exactly what Merge steps an LI will partake in, similar to any other theory of subcategorization.

A sequence of Merge steps can be represented as a *derivation tree* as in (1a), with the corresponding phrase structure tree shown in (1b). Each Merge step of the derivation is represented by an interior node labeled with •.

(1) a. **Derivation tree**



b. **Corresponding phrase structure tree**



Derivation trees provide a more abstract description of syntactic structure that focuses on the grammatical operations rather than their output, the *derived structures*. In the terminology of Chomsky (1986), they are a direct representation of I-language operations, not objects of E-language. A derivation tree acts as a common blueprint for all of the following: a canonical X′-tree (Jackendoff 1977), the compacted X′-tree in (1b), a bare phrase structure set (Chomsky 1995), a PF-structure

with prosodic modifications in the spirit of Richards (2016), a logical form, or simply the output string. Each one of these can be produced from the same derivation tree. For this reason, derivation trees provide a unified representation format and are the ideal measuring rod for the complexity of the grammar's operations *modulo* differences in output representations.

The next section explains how Minimalist operations can be reinterpreted as constraints on derivation trees, which in turn makes it possible to equate the complexity of operations with the complexity of constraints over derivation trees.

2.2 Merge as a constraint on derivation trees

As discussed at the beginning of §2.1, Merge is a feature-triggered operation in MGs. So a computational system that has to correctly apply Merge must ensure that no requirements of the feature calculus are violated. For Merge, this involves two factors:

(2) **Conditions on Merge**

a. For every LI l and selector feature X^+ of l, X^+ must be checked as part of an application of Merge.

b. Let t and u be two syntactic objects headed by h_t and h_u, respectively. Then t and u may be merged iff the first unchecked feature of h_t is some selector feature X^+ and the first unchecked feature of h_u is the category feature X^-.

These conditions on Merge can be translated into constraints on the shape of derivation trees. The definition of these constraints is fairly simple once a few key concepts have been put in place.

The first one is the notion of an *ancestor chain* of a node m. The full ancestor chain consists of all the nodes that are distinct from m and dominate m. The ancestor chain of length n consists of the n structurally lowest nodes in the full ancestor chain. In (1b), the full ancestor chain of *'s* is $\langle \mathrm{D}', \mathrm{DP}, \mathrm{VP} \rangle$, whereas the ancestor chain of length 2 is $\langle \mathrm{D}', \mathrm{DP} \rangle$.

Ancestor chains are the basis for another two formal terms, *host* and *D[erivational]-root*. Every LI with exactly n selector features ($n \geq 0$) is a host of itself and a host of every node in its ancestor chain of length n. Furthermore, if m is the highest node in l's ancestor chain of length $i \leq n$, then m is hosted by the i-th selector feature of l. The *D-root* of an LI l is the structurally highest node hosted by l. Intuitively, the D-root of an LI is the derivational counterpart to its maximal projection in the derived tree.

In (1a), *'s* is the host of itself, its mother and the mother of its mother; the root node, on the other hand, is hosted by *scolded*. More precisely, the root node is hosted by the second D^+ feature on *scolded*. The D-root of *'s* is the left daughter of the root node, and the root node is the D-root of *scolded*.

With the notions of ancestor chain, host, and D-root sufficiently formalized, Merge can be recast as the conjunction of three constraints on derivation trees.

(3) **Merge constraints on derivation trees**

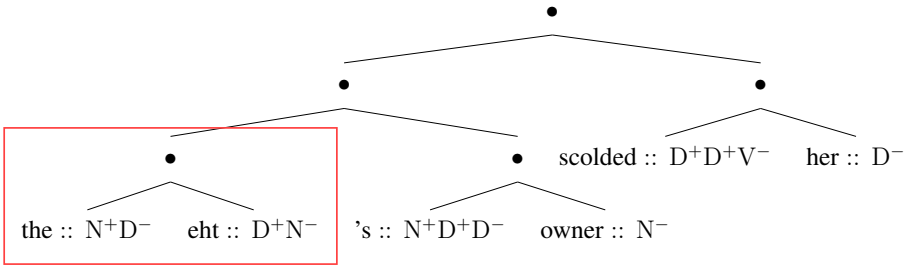
a. *Uniqueness*
Every Merge node is hosted by exactly one LI.

b. *Full selection*
Every LI with exactly n selector features hosts exactly n Merge nodes.

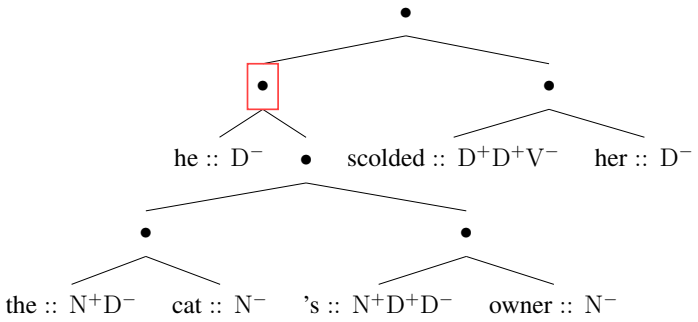
c. *Match*
Let m be a Merge node hosted by the selector feature X^+ on l. Then exactly one of its daughters must be the D-root of an LI with category feature X^-.

A derivation tree contains an illicit Merge application iff one of the constraints above is violated.

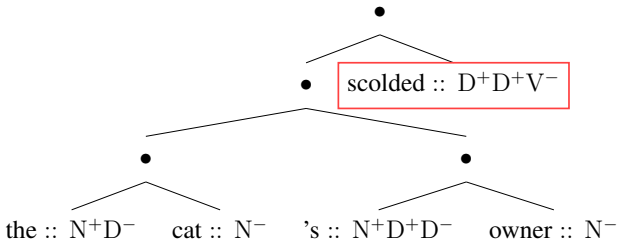
(4) a. **Violation of uniqueness (too many hosts)**



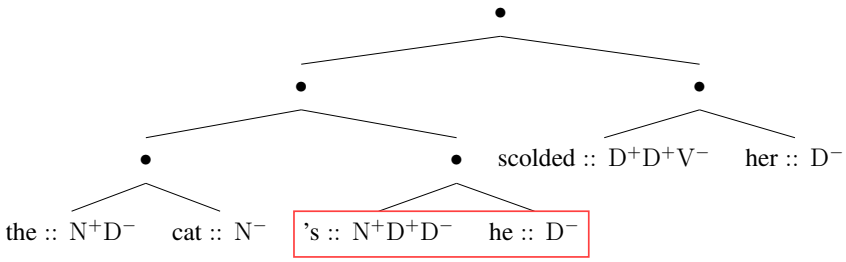
b. **Violation of uniqueness (no host)**



c. **Violation of full selection**



d. **Violation of match**



The representational, constraint-based view of the operation Merge allows us to assess the complexity of Merge in terms of a specific formal problem. Let Lex be a set of LIs annotated with selector and category features in the usual manner. Then there is a unique (and usually infinite) set of well-formed derivation trees that can be built from Lex. In formal language theory, such a set is called a *tree language*, just like a *string language* is a set of strings. With respect to Merge, the only criterion for well-formedness of a derivation tree is whether it obeys the constraints in (3). Therefore the complexity of Merge can be equated with the complexity of these derivation tree languages, which is readily established with the help of previous work from formal language theory.

2.3 Merge without adjunction is in SL

We are now in a position to evaluate the result in Graf (2012) that Merge belongs to the so-called class SL. First of all, the theorem can now be stated more precisely.

(5) **Complexity of Merge**
Let G be an MG whose only operation is Merge. Then the derivation tree language of G is in SL.

So it is not Merge itself that is in SL, but rather we use Merge-only derivation tree languages as a proxy for measuring the complexity of the operation that produces these languages.

SL is short for *strictly local*, and this name already indicates why Merge is in SL. In contrast to Move, Merge does not establish a dependency between two nodes that are arbitrarily far away from each other. Instead, selector and argument are very close to each other. The class SL is a mathematical formalization of this intuition.

For the sake of exposition, let us first consider a string-based example from phonology. The process of intervocalic voicing can be viewed as a constraint against voiceless consonants that are flanked by vowels. We may express this symbolically as a constraint $^{*}\mathrm{V}[-\mathrm{voice}]\mathrm{V}$. The string *bahazam* obeys this constraint, whereas *bahasam* violates it. But how would a computational device determine that the first string is well-formed and the second one ill-formed?

One simple solution is a scanner. Imagine a search window moving through each string from left to right. For our example, the window is only large enough to see three adjacent segments at a time. Each sequence of three adjacent segments is compared against the constraints imposed by the grammar. If at any given time the window contains a sequence that matches the pattern $^{*}\mathrm{V}[-\mathrm{voice}]\mathrm{V}$, the string is illicit. If, on the other hand, the search window can make it all the way to the end of the string without seeing even one forbidden configuration, the string is well-formed.

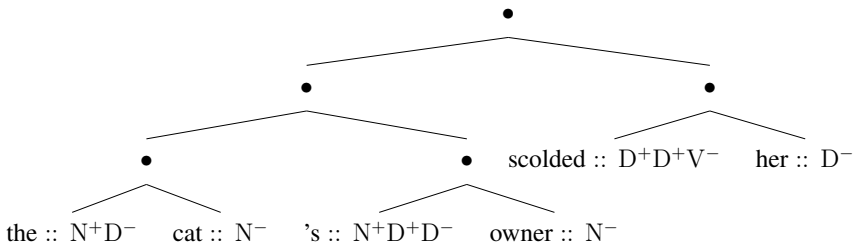
A string language is SL-k iff one can specify a finite number of constraints such that a scanner with a search window of size k can correctly determine for every string whether it is well-formed or ill-formed. A string language is SL iff it is SL-k for some k.

SL has mostly been explored for string languages (see Heinz 2015 and references therein), but it can be lifted from strings to trees. In this case, the search window moves through a tree instead of a string, and its length measures how deep

into the tree one gets to see at any point. For example, a search window of size 2 can only see a node and its daughters, whereas a window of size 3 can also see the daughters of the daughters. For Merge to be SL, there has to be some k for each MG (the value may differ between grammars) such that a search window of size k is sufficient to detect any potential violations of the constraints in (3).

Consider once more the example derivation from (1a), repeated here:

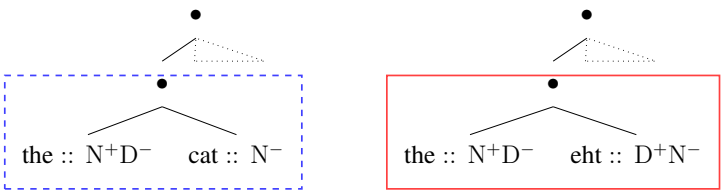
(1a′)



For the sake of exposition, we will assume that these LIs are representative of the whole grammar insofar as no LI has more than two selector features. Now let us look at how a search window of size 4 can correctly enforce all Merge constraints in (3) given such an upper limit.

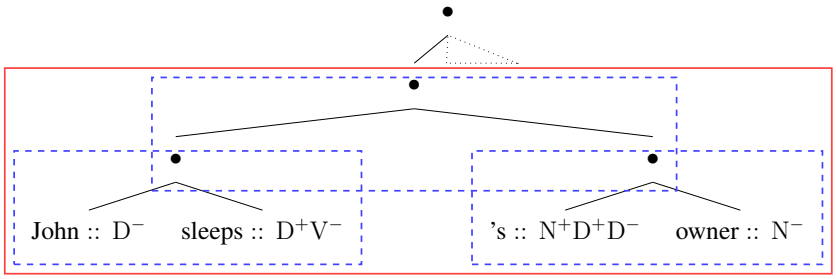
In order to determine that the merger of *the* and *cat* is licit, a window of size 2 would actually suffice. Within this window, we see that the Merge node is uniquely hosted, that full selection is satisfied for *the*, and that *cat* has a matching category feature. Hence a window of this size is also enough to block illicit configurations like the one we encountered in (4a).

(6)



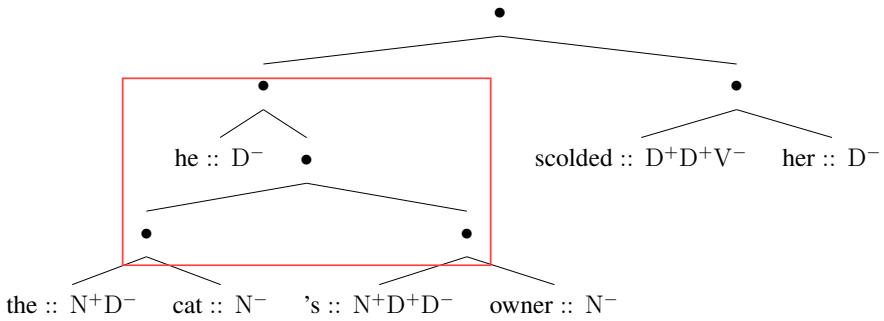
But a window of size 2 is not enough for all configurations. Verifying that all Merge requirements are met with respect to *'s* demands an increase of the search window size from 2 to 3. This is illustrated in (7), where no window of size 2 can detect that the second argument of the possessive marker is a VP rather than a DP. A window of size 3 does, though, because it can see both *'s* and *sleeps* at the same time.

(7)



A size 3 window will also notice the unhosted Merge node from (4b), as is shown below.

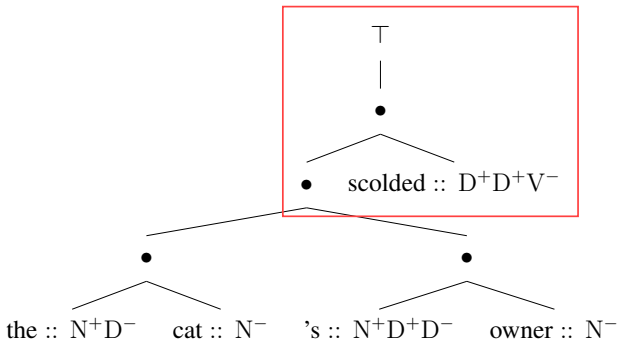
(8)



Since no LI has more than two selector features, every Merge node must be the mother of an LI with one selector feature or the mother of the mother of an LI with two selector features. Consequently, a search window of size 3 necessarily includes the host of the highest Merge node, if it has one. This is not the case in the tree above, which is thus inferred to be illicit.

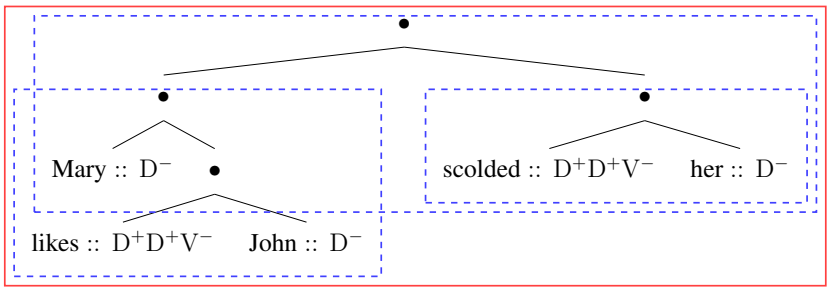
In addition, a size 3 window will recognize all full selection violations. Full selection requires an LI l with exactly n selector features to have an ancestor chain of length n. This can be violated only if the length of the full ancestor chain of l does not exceed $n - 1$. In our example, where no LI has more than two selector features, no LI with two selector features may have a full ancestor chain of length 1. Suppose that we use the special marker $\top$ to indicate the top edge of the tree. This node is not part of the tree, it is just a notional trick to highlight that a size 3 window can tell whether the mother of *scolded* has a mother, too. In order for full selection to be satisfied, no search window of size 3 may contain both $\top$ and an LI with two selector features.

(9)



A size 3 window thus rules out many illicit Merge configurations for our example grammar, but still not all of them. Only a window of size 4 can correctly evaluate the Merge dependency that holds between *scolded* and the head of its specifier. The reason is exactly parallel to what we saw in (7) for *'s* and its specifier, except that the head of the specifier selects two arguments now instead of one, which increases its distance to *'s*.

(10)



Crucially, though, the search window does not need to grow past size 4 for this grammar. The reader is invited to verify this for themselves: no matter how large the derivation tree, every violation of a Merge constraint can be detected within a window of size 4 as long as no LI selects more than two arguments. In fact, it holds for every MG that the size of the search window never needs to exceed $k + 2$, where k is the maximum number of selector features on a single LI. It is thanks to this tight link between window size and maximum number of arguments that the complexity of Merge stays within the class SL.

3 Merge with adjunction: from SL to TSL

Let us take stock of the findings so far: The complexity of Merge has been characterized in terms of the complexity of derivation tree languages that contain all trees, and only those, that obey all the Merge-related constraints in (3). Since Merge is a local relation, all these constraints can be enforced within a small search window of bounded size. This confirms the theorem of Graf (2012) that Merge belongs to SL, one of the weakest known classes of formal languages.

But the locality of Merge is not an intrinsic property of the operation, it depends on the grammar as a whole and can be disrupted by other operations. This is exactly what we will observe in this section. Once an adjunction operation is added to MGs, the distance between a selector and the head of its argument can grow without bounds due to the unlimited iterability of adjuncts. This is not necessarily a problem, as there are many different ways to implement adjunction (§3.1), and many of those still allow for Merge to be regulated in a local fashion as long as one can only adjoin to arguments (§3.2). However, once it becomes possible to adjoin to an adjunct, i.e. once adjunction may apply recursively, Merge is no longer SL (§3.3). Instead, it is pushed into the more complex class TSL (§3.4). As mentioned earlier and as will be discussed in detail in §4, Move also belongs to TSL. Therefore the subregular complexity of a grammar with Merge and adjunction is not increased by the addition of Move.

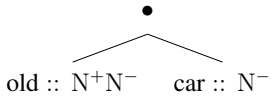
3.1 Models of adjunction

There are numerous MG implementations of adjunction, some of which are more complex than others (see Fowlie 2013 and Graf 2014 for further details). In order to make the computational argument that Move is no more complex than Merge as strong as possible, I will adopt an implementation in §3.2 that keeps the complexity increase for Merge fairly minimal. Other formalizations of adjunction are so intricate that they push Merge far beyond TSL. Quite generally, the more information is

encoded directly in the adjuncts via features, the smaller the increase in subregular complexity for Merge.

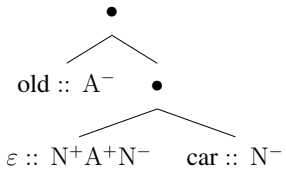
That said, there is one implementation that is even simpler than the one I will adopt. This variant reduces adjunction to a special case of selection and thus does not alter the complexity of Merge at all. For example, *old* would not be treated as an adjective that adjoins to the noun *car*, but as a noun that selects *car*.

(11) **Adjunction as category-preserving selection**



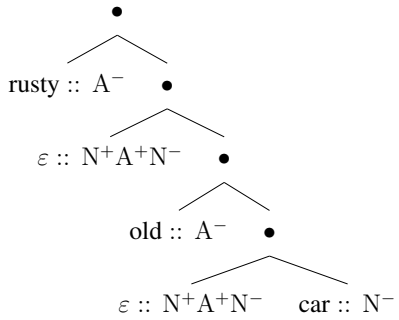
It seems strange to treat *old* as a noun, though, so a more elaborate version might instead posit an empty head ε :: $\mathrm{N^+A^+N^-}$ which implicitly converts an adjective into an NP-adjunct.

(12) **Adjunction as category-preserving selection by an empty head**

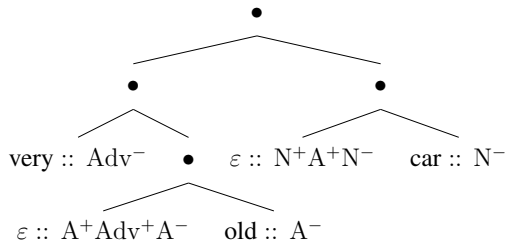


This analysis still allows for an unlimited number of adjuncts, and it is also possible to adjoin to an adjunct.

(13) a. **Multiple adjunctions**



b. **Recursive adjunction**



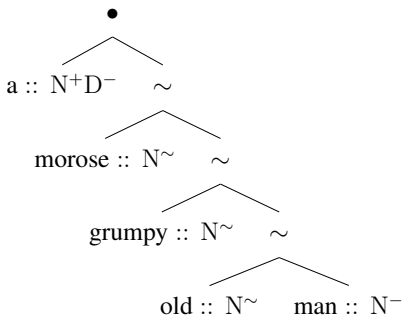
Even though this analysis is appealing in its simplicity, its conflation of arguments and adjuncts is also a detriment. Without further stipulations, this view predicts that arguments and adjuncts should behave the same. Yet there is plenty of

empirical evidence to the contrary. For example, arguments are usually easier to extract (and extract from) than adjuncts, and pronouns within certain adjuncts seem to be less restricted with respect to their choice of binders. Such differences are less surprising if adjunction is a separate operation in the grammar, as I will assume for the remainder of the paper.

3.2 Merge with non-recursive adjunction is SL

Frey & Gärtner (2002) propose to model adjuncts as LIs that have an adjunction feature $\mathrm{X}^{\sim}$ instead of a category feature Y^{-}. The adjunction feature requires an LI to adjoin to another LI l with category feature X^{-}, provided that l has already selected all its arguments. The category feature X^{-} of l is not checked by any of the adjuncts, which permits repeated adjunction to the same phrase.

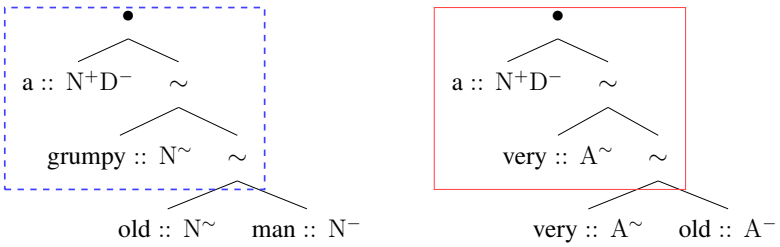
(14)



Without an upper bound on the number of adjuncts per phrase, Merge dependencies become unbounded. In the example above, more and more adjuncts can be added to increase the distance between *a* and *man* by any desired amount. As a result, there is no k such that a window of this size is guaranteed to contain both LIs.

This does not imply, though, that Merge is no longer SL. The only reason why one might want to fit *a* and *man* into the same search window is to verify that the match constraint of (3) is still obeyed: N^{+} on *a* must have a matching counterpart N^{-} on *man*. But that can also be determined indirectly by looking at the adjunction feature $\mathrm{N}^{\sim}$. Since an LI with feature $\mathrm{N}^{\sim}$ requires the presence of an LI with category feature N^{-}, the presence of the latter can be inferred from the former.

(15)

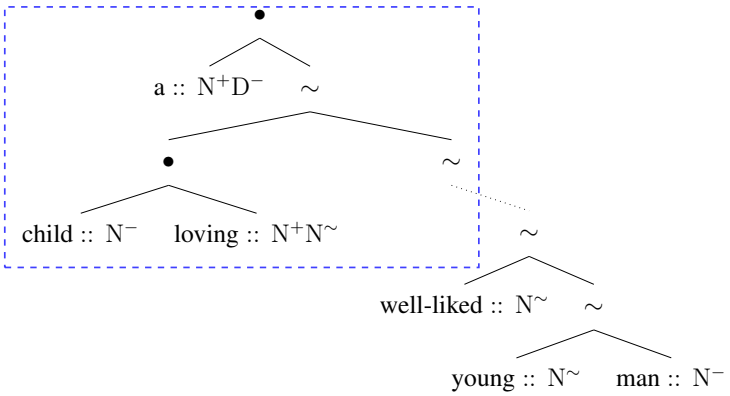


Of course this presupposes that no derivation tree contains illicit adjunction steps, e.g. an NP-adjunct that adjoins to a VP or, even worse, to nothing at all. This assumption is entirely innocent for the purposes of this paper because the focus is on the subregular complexity of Merge, not adjunction. How exactly adjunction is

to be regulated via constraints on derivation trees is orthogonal to the paper's goals and will be put aside here. At this point, the important insight is that Merge is still SL in the original adjunction system of Frey & Gärtner (2002).

That Merge is SL holds even if adjuncts may select arguments of their own. As long as no adjunct may undergo adjunction before all its selector features have been checked, a fixed-size window is still sufficient because each adjunct can only select a bounded number of arguments.

(16)

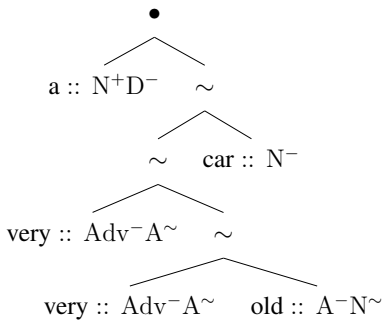


While adjuncts may take arguments, the current system makes it impossible for them to contain adjuncts of their own — adjunction can only target phrases whose head has a category feature, but category features and adjunction features are mutually exclusive. Next we will see that relaxing this condition pushes Merge beyond SL.

3.3 Merge with recursive adjunction is not SL

Suppose that adjuncts still carry a category feature, which is immediately followed by an adjunction feature. For example, the feature string of *old* would now be expanded from N∼ to A⁻N∼. Since category features are the only prerequisite for attracting adjuncts, this system allows for adjunction to an adjunct.

(17) **Recursive adjunction**



This minor change has major repercussions as Merge is no longer guaranteed to be in SL. As in §3.2, the head h and its selectee s might be separated by an arbitrary number of adjuncts $a_1, a_2, \ldots, a_m$. But now each a_i may have an arbitrary number of adjuncts $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ of its own, so that each a_i is also arbitrarily far away

from h. Consequently, the search window does not necessarily contain the adjunct feature of any a_i, which one could infer the category feature of s from.

One might attempt to infer the adjunction feature of a_i from some a_{i_j}, mirroring our earlier strategy to infer the category of s from a nearby adjunction feature. But recursive adjunction makes this impossible, too. Each adjunct a_{i_j} of adjunct a_i may also have an arbitrary number of adjuncts, so that a_{i_j} might not be part of the search window either. Given a sufficiently large number of recursive adjunction steps, a search window containing the head h will only contain the Merge node above h and a humongous number of adjunction nodes labeled $\sim$, but none of the adjuncts themselves. Without any other feature-carrying nodes in the search window, there is not enough information to determine indirectly whether the selectee s has a matching category feature for the relevant selector feature on h.

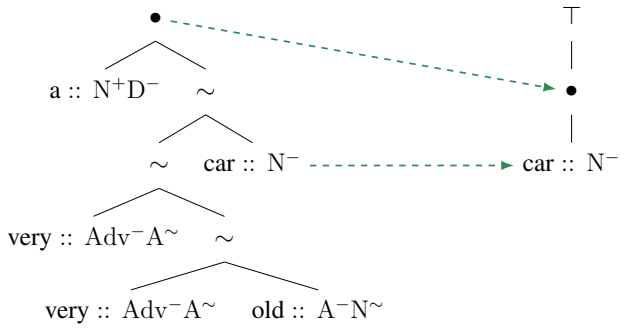
At a more abstract level, the challenge posed by recursive adjunction compared to its non-recursive counterpart is the increased difficulty in decomposing a non-local dependency into a local one. With non-recursive adjunction, the non-local Merge dependency is instead mediated by local dependencies between h and a_1, a_1 and a_2, a_2 and a_3, ..., and finally a_m and s. With recursive adjunction, this decomposition is not helpful because each one of these dependencies may still be non-local, given that each a_i may have an unbounded number of adjuncts of its own. These adjuncts of the adjuncts may also have adjuncts, so that the result of decomposing their non-local dependencies may again be a collection of non-local dependencies. The recursive nature of adjunction entails that there is no upper bound on how often one has to decompose non-local dependencies to end up with local ones, and this is why Merge is pushed out of SL.

Although Merge is no longer SL in a grammar with recursive adjunction, that does not mean that it does not display any degree of locality. The next section shows that if one has a mechanism to mask out irrelevant parts of a derivation tree, Merge is once again SL. The combination of SL with such a masking mechanism is the core idea behind TSL.

3.4 Merge with recursive adjunction is TSL

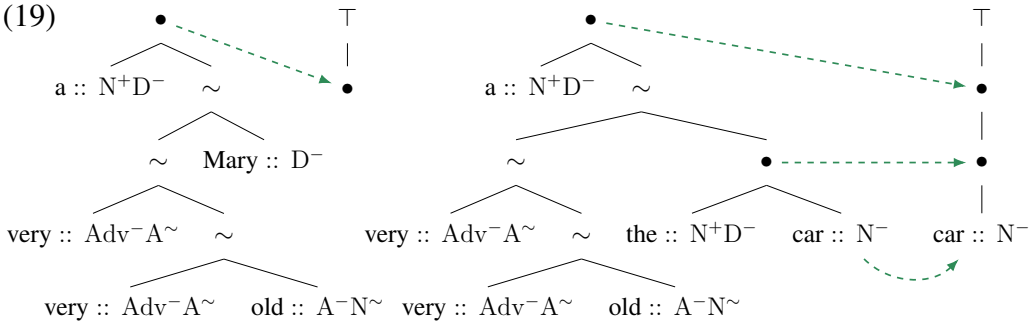
Suppose that we take the derivation tree in (17) and remove all nodes except those that either are hosted by a selector feature X^+ or carry a category feature X^- but no adjunction features. Let us call the structure produced by this removal step the *X-tier* that is projected from the derivation tree. Example (18) illustrates the construction of an N-tier, which contains all Merge nodes hosted by N^+, all non-adjunct LIs with N^-, and nothing else. For mathematical convenience, we also add a dedicated root node $\top$ to ensure that a tier is always a tree (cf. (20)). The presence of $\top$ will also be useful in §4.

(18) **Constructing an N-tier**



Now contrast (18) with the N-tiers projected from ill-formed derivations.

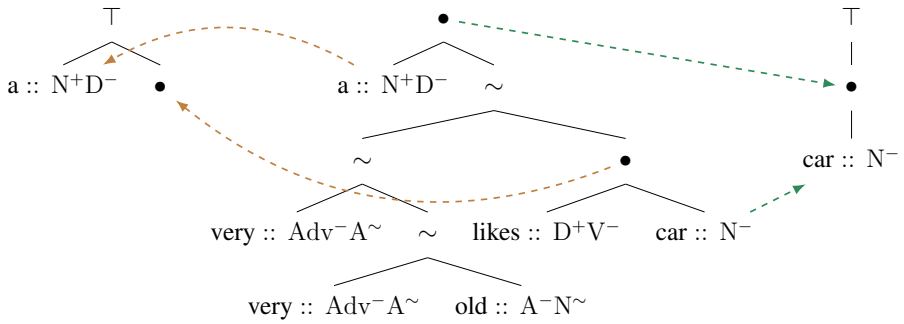
(19)



In both cases, the tier contains a Merge node that does not have an LI among its daughters. From the peculiar shape of these projected N-tiers one can immediately infer that the match constraint from (3) is violated in the corresponding derivation trees.

Note that the inverse is not necessarily true: an N-tier where every Merge node has exactly one LI among its daughters may still have been projected from an illicit derivation tree. But then some other tier, e.g. the D-tier, will be ill-formed.

(20)



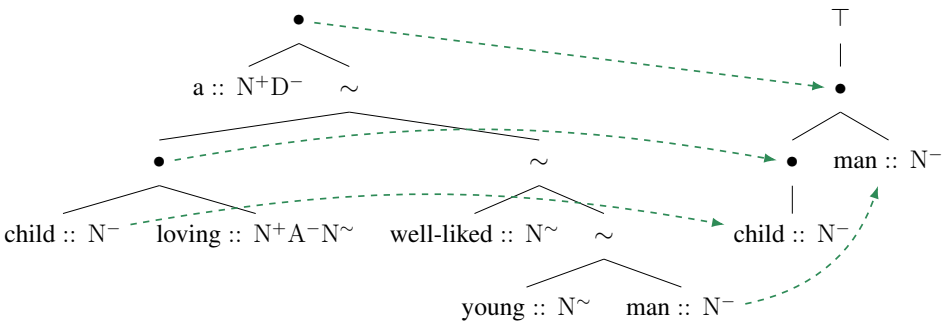
Example (20) shows that if one projects a tier for each kind of category/selector feature, then any violations of the match constraint will surface on these tiers in the form of a Merge node without an LI as a daughter.

(21) **Match over tiers**
No tier must contain a Merge node without exactly one LI among its daughters.[1]

[1]For the sake of exposition, I make the simplifying assumption that no LI contains both a cate-

The constraint indirectly enforces the match condition on Merge as a condition on the shape of Merge tiers. It yields the desired result even in configurations where a Merge node has multiple daughters on a tier.

(22)



While there is not enough room to flesh out this sketch into a formal proof, it should be clear that even though the match constraint on Merge is not local over derivation trees, it is local over the Merge tiers that are projected from these derivations. But what about the other two constraints in (3), uniqueness and full selection? These constraints are in fact SL even over derivation trees with recursive adjunction. This is because they regulate how Merge nodes are to be hosted by LIs with selector features. Since adjunction can only target LIs that have already had all their selector features checked, no adjunction node can occur between a Merge node and the LI that hosts it. So adjunction does not disrupt the locality of these dependencies, and thus they are still in SL. The challenges of adjunction are all limited to the match constraint; as we just saw, it is still local over the right kind of structure.

Let me briefly summarize the findings on Merge in MGs with adjunction. As in MGs without adjunction, Merge is a combination of three constraints on derivation trees: uniqueness, full selection, and match. The first two are SL irrespective of whether the grammar has an adjunction operation. The latter, match, is SL only if adjunction cannot apply recursively. Otherwise, it requires the projection of Merge tiers for every category/selector feature X — over these tiers, match reduces to the requirement that every Merge node must have exactly one LI among its daughters.

This combination of tier projection and local conditions on these tiers is the hallmark of the subregular class TSL (Heinz *et al.* 2011). Just like SL, TSL was originally defined for string languages rather than tree languages. Lifting it from strings to trees is slightly more complex than for SL. Without going too much into technical details, one has to specify two components:

(23) **Components of TSL definition for trees**

a. The *projection function* determines which nodes are ignored and which are added to the tier. In standard TSL, the projection function makes this decision based solely on the label of the node, but more powerful versions also allow local context to be taken into account (De Santo & Graf 2017). Merge requires such a structure-sensitive tier projection to determine what feature a Merge node is hosted by.

gory feature X^- and a selector feature X^+. Such LIs introduce several technical complications in how the constraints must be stated, but do not affect the overall subregular complexity of Merge.

b. The *licensing function* maps each node in the tier to a language L of permitted strings. The tier is illicit if the node's string of daughters is not a member of L. Again this function may consider only the label of the node or take its structural context into account.
For Merge, the licensing function only considers the node label. It maps each $\bullet$ to the language $\bullet^* l \bullet^*$, where $\bullet^*$ denotes 0 or more instances of $\bullet$, and l is an LI. LIs are mapped to the empty set by the licensing function, which encodes that they must be leaf nodes.

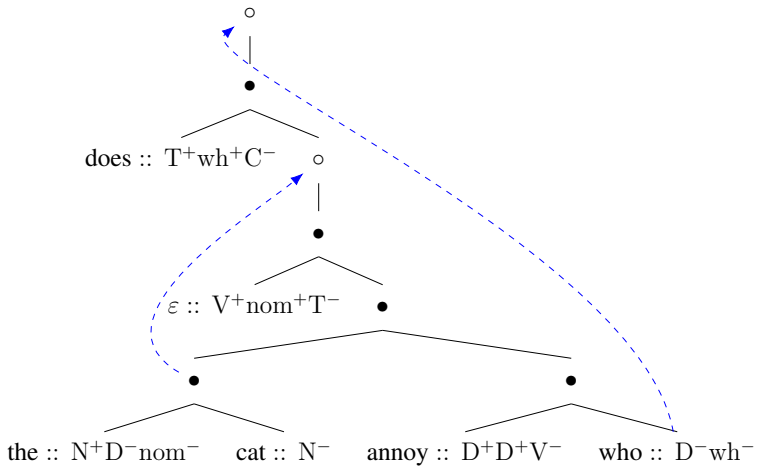
Within the class of TSL, Merge is actually fairly simple. While its tier projection function has to take a limited amount of context into account, the licensing function only considers node labels. In addition, the string language $\bullet^* l \bullet^*$ is very simple — it is TSL-2 in the sense of Heinz *et al.* (2011) and is a syntactic counterpart to well-known phonological phenomena. For example, the requirement that every phonological word has exactly one primary stress corresponds to the formal language $\sigma^* \acute{\sigma} \sigma^*$, where σ denotes an unstressed syllable and $\acute{\sigma}$ a stressed one. The parallels between $\bullet^* l \bullet^*$ and $\sigma^* \acute{\sigma} \sigma^*$ are evident. In sum, the subregular complexity of Merge is still very limited even though it has been pushed from SL to TSL.

It was already mentioned at the very beginning of the paper that Move, given certain assumptions, is also TSL. However, the parallels between Merge and Move extend far beyond that. As we will see next, Move tiers are constructed and regulated in exactly the same fashion as Merge tiers, which reveals an enormous amount of parallelism between the two operations.

4 Move is in TSL, too

The MG operation Move is modeled closely after the Minimalist notion of movement, but it looks very different through the lens of derivation trees. In a derivation tree, no subtrees are ever moved into a different position — the actual displacement only happens during the construction of the derived structure. As such, Move is merely a node in a derivation tree whose distribution is controlled by the presence of certain features on LIs. Whereas Merge is triggered by the presence of both a selector feature X^+ and a category feature X^-, Move requires a *licensor feature* f^+ and a *licensee feature* f^-. This is illustrated below for a derivation with wh-movement and case movement, where $\circ$ represents Move and arrows are used as an expository device to highlight what moves where.

(24) **Derivation tree with wh-movement and case movement**



Note that the arrows have been added only for the reader's convenience as the MG feature calculus is sufficient to determine what moves where. Once the feature annotations of all LIs in a derivation are known, Move is fully deterministic in MGs. This may seem like a marked departure from Minimalist syntax, where occasionally multiple phrases compete for the right to move to a given position. Such ambiguities are captured in MGs by allowing different feature annotations for one and the same LI, so the difference between MGs and Minimalism is again smaller than it seems.

Another important difference is the status of intermediate movement in the feature calculus. Graf *et al.* (2016) — generalizing a result of Kobele (2006) — prove that every MG can be brought into a normal form where intermediate movement is no longer triggered by features. That is to say, every LI l carries at most one licensee feature and the only feature trigger for movement is at the final landing site of l. Any intermediate landing sites are automatically inserted during the construction of the derived structure. For example, a wh-phrase moving from Spec,vP to Spec,CP automatically leaves a trace/copy in Spec,TP even though no feature checking of nom^+ and nom^- takes place. This result isn't just a technical curiosity (see Graf 2018 for a detailed defense); Graf & Heinz (2015) show that it is essential in bringing out the TSL-nature of Move.

Move can be decomposed into two constraints over derivation trees. Both rely on the notion of *occurrence*. Given an LI l with licensee feature f^-, its occurrence is the structurally lowest Move node that dominates l and is hosted by a feature f^+. In (24), the higher Move node is the occurrence of *who* and the lower one the occurrence of *the*. A derivation tree is well-formed with respect to Move if three conditions are satisfied.

(25) **Conditions on Move**

a. Every LI with n licensor features hosts exactly n Move nodes.

b. Every LI with a licensee feature has an occurrence.

c. Every Move node is an occurrence of exactly one LI.

These constraints are satisfied by (24). But suppose that *who* carried nom^- instead of wh^-, creating an illicit configuration. The higher Move node would not be an occurrence of any LI, whereas the lower one would be an occurrence of two LIs.

Similarly, if the higher Move node were missing in (24), then the C-head would not be hosting any Move nodes even though it has licensor features, and *who* would have a licensee feature yet lack an occurrence. Any other illicit configurations are just variations of these few base cases, so that the constraints in (25) are indeed sufficient for regulating Move.

It is easy to see that Move cannot be SL because there is no upper bound on the distance between an LI and its occurrence. This does not affect (25a), since every Move node is only a fixed distance away from its host. But (25b) and (25c), which establish dependencies between LIs and their occurrences, cannot be SL for this reason. Just as for Merge, though, the relevant constraints are local over tiers.

For each movement type f, one projects a tier that contains I) all Move nodes that are hosted by some f^+, and II) all LIs with licensee feature f^-, and III) nothing else. Over these tiers, two constraints apply.

(26) **Move as constraints over tiers**

a. Every Move node has exactly one LI among its daughters.

b. Every LI has a Move node as its mother.

From the perspective of TSL, (26a) maintains that every Move node must have a string of daughters that fits the pattern $\circ^* l \circ^*$, closely mirroring the language $\bullet^* l \bullet^*$ for Merge. Constraint (26b) is recast in TSL-terms with the help of the assumed root node $\top$: its string of daughters must fit the pattern $\circ^*$.

No constraint like (26b) was mentioned during our discussion of Merge in §3.4, but it nonetheless has a natural counterpart. It is commonly assumed that only CPs can be well-formed sentences, whereas TPs or DPs are too small and thus must be merged with something else. Consequently, only the C-tier may contain an LI that does not have a Merge node as its mother. So N-tiers, D-tiers, and so on, are exactly like move tiers in that the root $\top$ must have a string of daughters of the form $\bullet^*$.

This is but a brief sketch of the TSL-nature of Move, with many technical details meriting extended discussion. Due to space constraints, I unfortunately have to refer the reader to Graf (2012), Graf & Heinz (2015), Graf *et al.* (2016), and Graf (2018). For the purposes of this paper, though, the important issue is the pronounced parallelism between Merge and Move with respect to subregular complexity. Not only do they both fall into the class TSL, they are remarkably similar regarding how tiers are projected and regulated.

(27) **TSL-comparison of Merge and Move**

	Merge	**Move**
Projection	LI with X^-	LI with f^-
	$\bullet$ if hosted by X^+	$\circ$ if hosted by f^-
Licensing	$\bullet \mapsto \bullet^* l \bullet^*$	$\circ \mapsto \circ^* l \circ^*$
	$\top \mapsto \bullet^*$(or l for C-tier)	$\top \mapsto \circ^*$

5 Conclusion

Merge and Move have been argued to be very distinct regarding their subregular complexity as the former falls into the class SL, whereas the latter belongs to the

strictly more powerful class TSL. I have shown that this difference disappears in grammars with recursive adjunction, where Merge is also TSL. A subregular view rooted in derivation trees thus reveals Move to be a natural extension of Merge, mirroring the original argument of Chomsky (2004).

Several issues had to be left open. Most importantly, I only compared the operations in terms of their derivational behavior, rather than their effect on the externalization function that produces the derived trees according to the derivational blueprint. In that respect, Move is still more complex than Merge and adjunction because it requires the controlled displacement of substructures, which is a computationally demanding process.

In addition, the subregular complexity of adjunction is still unknown. The same holds for Merge in a system where adjunction is not controlled by features (cf. Graf 2014), and Move if intermediate movement is also triggered by features. In the spirit of this paper, though, I expect that these more complex systems will again display a large degree of parallelism.

References

Aksënova, A., T. Graf, & S. Moradi. 2016. Morphotactics as tier-based strictly local dependencies. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, 121–130.

Chandlee, J. *Strictly Local Phonological Processes*. University of Delaware dissertation.

Chomsky, N. 1986. *Knowledge of Language: Its Nature, Origin, and Use*. New York: Praeger.

Chomsky, N. 1995. Bare phrase structure. In *Government and Binding Theory and the Minimalist Program*, ed. by G. Webelhuth, 383–440. Oxford: Blackwell.

Chomsky, N. 2004. Beyond explanatory adequacy. In *Structures and Beyond: The Cartography of Syntactic Structures Volume 3*, ed. by A. Belletti, 104–131. Oxford: Oxford University Press.

De Santo, A., & T. Graf, 2017. Structure sensitive tier projection: Applications and formal properties. Ms., Stony Brook University.

Fowlie, M. 2013. Order and optionality: Minimalist grammars with adjunction. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, ed. by A. Kornai & M. Kuhlmann, 12–20.

Frey, W., & H.-M. Gärtner. 2002. On the treatment of scrambling and adjunction in Minimalist grammars. In *Proceedings of the Conference on Formal Grammar*, ed. by G. Jäger, P. Monachesi, G. Penn, & S. Wintner, 41–52.

Graf, T. 2012. Locality and the complexity of Minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. by P. de Groot & M.-J. Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227, Heidelberg. Springer.

Graf, T. 2014. Models of adjunction in Minimalist grammars. In *Formal Grammar 2014*, ed. by G. Morrill, R. Muskens, R. Osswald, & F. Richter, volume 8612 of *Lecture Notes in Computer Science*, 52–68, Heidelberg. Springer.

Graf, T. 2017. The power of locality domains in phonology. *Phonology* 34.385–405.

Graf, T. 2018. Grammar size and quantitative restrictions on movement. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, 23–33.

Graf, T., A. Aksënova, & A. De Santo. 2016. A single movement normal form for Minimalist grammars. In *Formal Grammar : 20th and 21st International Conferences, FG 2015, Barcelona, Spain, August 2015, Revised Selected Papers. FG 2016, Bozen, Italy, August 2016*, ed. by A. Foret, G. Morrill, R. Muskens, R. Osswald, & S. Pogodalla, 200–215, Berlin, Heidelberg. Springer.

Graf, T., & J. Heinz. 2015. Commonality in disparity: The computational view of syntax and phonology. Slides of a talk given at GLOW 2015, April 18, Paris, France.

Heinz, J. 2009. On the role of locality in learning stress patterns. *Phonology* 26.303–351.

Heinz, J., 2015. The computational nature of phonological generalizations. Ms., University of Delaware.

Heinz, J., & W. Idsardi. 2013. What complexity differences reveal about domains in language. *Topics in Cognitive Science* 5.111–131.

Heinz, J., C. Rawal, & H. G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 58–64.

Jackendoff, R. 1977. *X-Bar Syntax: A Study of Phrase Structure*. Cambridge, MA: MIT Press.

Jardine, A. 2016. Computationally, tone is different. *Phonology* 33.247–283.

Kobele, G. M. *Generating Copies: An Investigation into Structural Identity in Language and Grammar*. UCLA dissertation.

Richards, N. 2016. *Contiguity Theory*. Cambridge, MA: MIT Press.

Stabler, E. P. 1997. Derivational Minimalism. In *Logical Aspects of Computational Linguistics*, ed. by C. Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.

Stabler, E. P. 2011. Computational perspectives on Minimalism. In *Oxford Handbook of Linguistic Minimalism*, ed. by C. Boeckx, 617–643. Oxford: Oxford University Press.