This is an expanded version of my paper for the CLS 2017 proceedings.
See v1 for the original, published version.
A changelog is attached at the very end.

Why movement comes for free once you have adjunction

Thomas Graf
mail@thomasgraf.net

April 26, 2018

Abstract

Based on a formal analysis of the operations Merge and Move, I provide a computational answer to the question why Move might be an integral part of language. The answer is rooted in the framework of subregular complexity, which has had great success in computational phonology. The subregular perspective reveals that Merge belongs to the formal class TSL 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 little computational or conceptual cost attached to it and comes essentially for free in any grammar with Merge and recursive adjunction.

1 Introduction

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 parts of a sentence are sometimes 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, at least in phonology and morphology (Heinz 2009; Heinz and Idsardi 2013; Chandlee 2014; Jardine 2016; Aksënova et al. 2016; Graf 2017b; Chandlee and Heinz 2018; a.o.). While subregular complexity has mostly been defined for string-based systems, it can be extended to trees and thus to syntax. The subregular view of syntax reveals a pronounced difference between Merge and Move.

Graf (2012) and Graf and 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 *tier-based strictly local* (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 at least 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. This challenges the SL-nature of Merge. As long as adjunction is not recursive (i.e. no adjunct can have adjuncts of its own), certain tricks can be used to keep Merge in SL. But no such tricks are available if adjuncts can also be adjoined to. As a result, Merge moves beyond SL into the class TSL, which is also occupied by Move under certain technical assumptions. It follows that Merge and Move are of comparable complexity. But the similarity does not stop there: upon closer inspection, the respective computations that underpin Merge and Move are almost exactly the same. Therefore, Move can indeed be regarded as a simple and very natural generalization of Merge.

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, §3.5). In §4, I then contrast the TSL-view of Merge against the TSL-view of Move as developed in Graf and 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. The implications of this finding are explored further in §5.

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 and how derivation trees provide a linguistically faithful representation of syntactic structure (§2.1). This also allows for a reanalysis of Merge as a bundle of constraints on derivation trees (§2.2). I then explain why Merge (in MGs without adjunction) is a locally bounded dependency and thus belongs to the very simple class SL (§2.3). I also point out a few other constraints on MG derivation trees that are indirectly related to Merge and belong to SL (§2.4). These results provide the backdrop for the subsequent discussion of Merge in MGs with adjunction (§3).

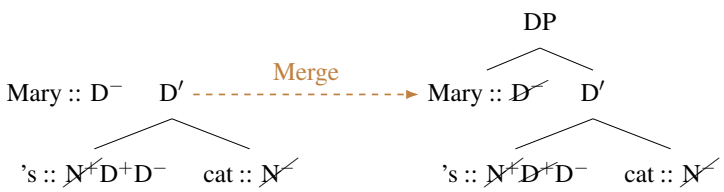
2.1 Merge in Minimalist grammars

Following MG tradition, but contrary to some recent proposals in Minimalist syntax, I assume that Merge does not apply freely but is mediated by a feature-driven subcategorization mechanism. That is to say, every lexical item (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.

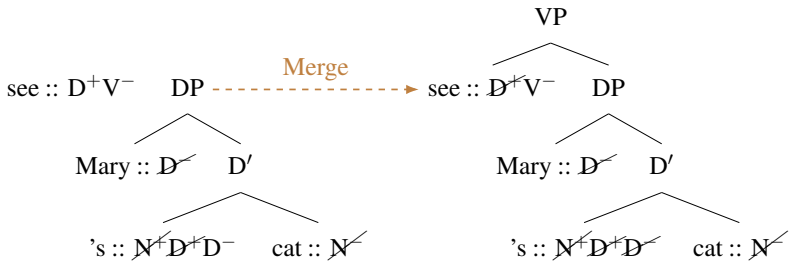
For example, the noun *cat* has only one feature, the category feature N^-. Therefore it can be selected by any LI that is looking for a noun, but it cannot take any arguments of its own. The determiner *a*, on the other hand, has the selector feature N^+ and the category feature D^-. More precisely, *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. The corresponding sequence of Merge steps is depicted below with checked features crossed out:

(1) a.

b.



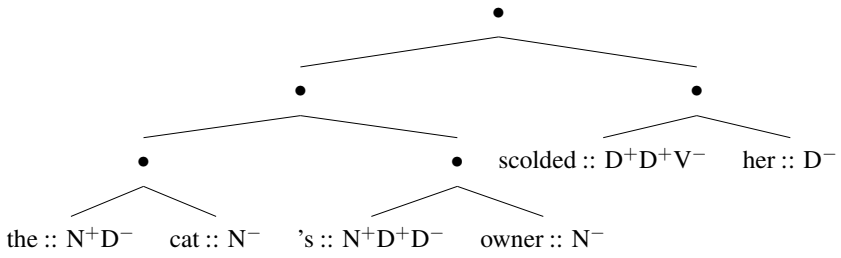
c.



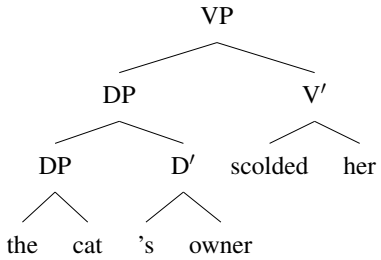
While feature ordering is never explicitly assumed in Minimalist syntax, there is an implicit consensus that whatever mediates selection exercises tight control over the order of arguments. This is why, say, *v* selects a VP as its complement and a subject DP as its specifier, rather than the other way 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 (2a), with the corresponding phrase structure tree shown in (2b). Each Merge step of the derivation is represented by an interior node labeled with •. For sake of exposition, the subject is selected by V instead of *v*.

(2) 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 (2b), 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.

It is also worth mentioning that derivation trees satisfy many Minimalist desiderata for syntactic representations (Chomsky 1995): they obey the extension condition and inclusiveness, and they are effectively unordered. The extension condition is necessarily observed because adding an operation O anywhere but at the root of the derivation tree would mean that O takes place before some of the operations that have taken place before the addition of O. This kind of "derivational time travel" is nonsensical. Inclusiveness holds because derivation trees are never altered by any operations, they are just a record of how some feature-annotated LIs were combined by the structure-building operations. Even feature checking does not result in removal of the checked features from the derivation tree, as this would destroy the record of the LIs' feature annotations. The linear order of siblings is irrelevant because it does not contribute any information that cannot already be inferred from the features of the LIs. By convention, derivation trees are drawn in a way that mirrors the order in the phrase structure tree, but no particular significance should be attached to this. The output of Merge(X,Y) depends solely on which one of the two, X or Y, contributes the selector feature. Consequently, Merge(X,Y) is always equivalent to Merge(Y,X). Overall, then, derivation trees are an excellent data structure for Minimalism because they I) provide a direct encoding of the I-language mechanisms rather than their E-language output, and II) satisfy many Minimalist desiderata for syntactic representations.

The remainder of this paper operates under the assumption that derivation trees are an abstract representation of the actual computations carried out by syntax (cf. Steedman 2001). Consequently, the complexity of derivation trees is indicative of the complexity of syntactic computations. The next section explains how this perspective allows us to reinterpret Minimalist operations as constraints on derivation trees, which in turn makes it possible to measure the complexity of syntactic operations in terms of the complexity of the corresponding constraints on 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:

(3) **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 (2b), the full ancestor chain of *'s* is $\langle \mathrm{D}', \mathrm{DP}, \mathrm{VP} \rangle$, whereas the ancestor chain of length 2 is $\langle \mathrm{D}', \mathrm{DP} \rangle$. The notion of ancestor chains can be applied to any arbitrary tree, including derivation trees.

Ancestor chains are the basis for another two formal terms, *host* and *D[erivational]-root*. D-root is defined in terms of host, which is turn builds on the notion of *positive* features. So far, the only positive features we have encountered are selector features like N^+. Category features like N^- are an instance of *negative* features. As we will encounter other types of positive and negative features during the discussion of Move in §4, I immediately specify host in terms of positive features rather than just selector features. Every LI with exactly n positive 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 positive 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 (2a), *'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, it 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.

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

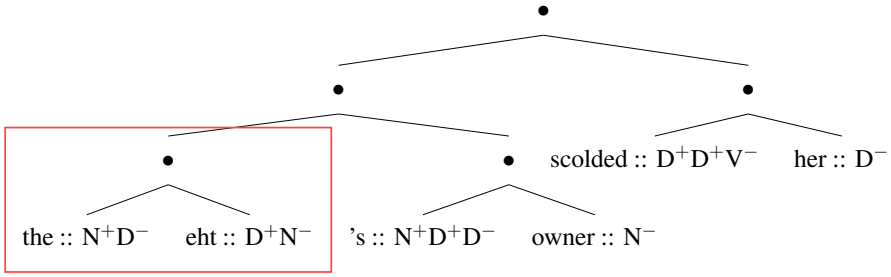
a. *Single head*
Every Merge node is hosted by exactly one LI.

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

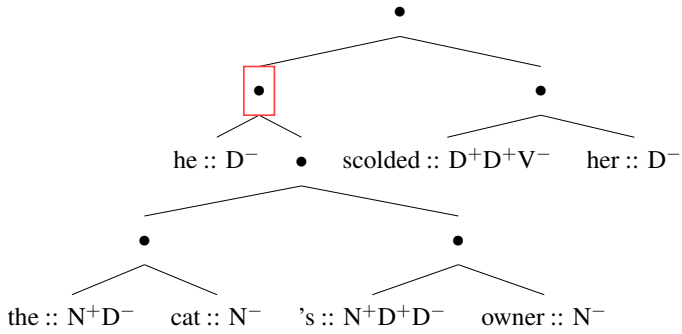
c. *Match*
Let m be a Merge node hosted by the selector feature X^+ on l. Then exactly one of the daughters of m must be the D-root of an LI with category feature X^-.

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

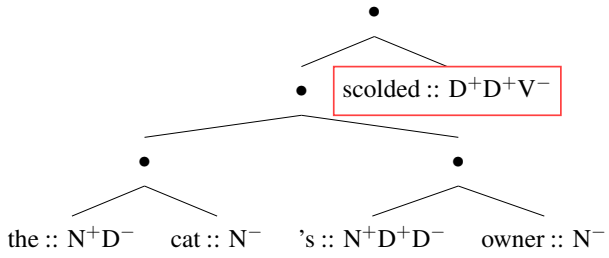
(5) a. **Violation of single head (too many hosts)**



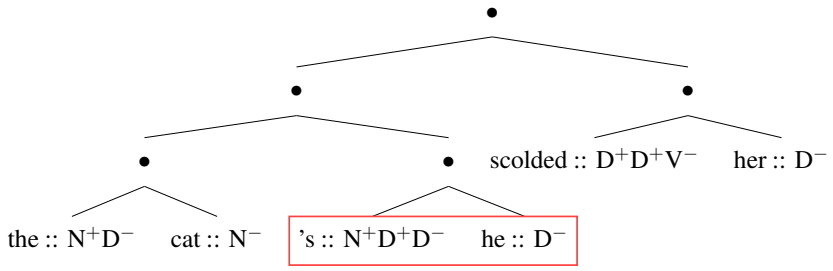
b. **Violation of single head (no host)**



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 (4). 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.

(6) **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 *balazam* obeys this constraint, whereas *balasam* violates it. But how would a computational device determine that the first string is well-formed and the second one ill-formed?

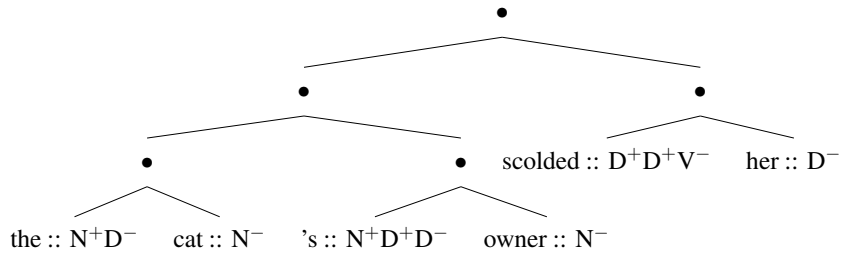
One simple solution is a scanner. Imagine a search window moving through each string from left to right. For our example, the window is only large enough to see three adjacent segments at a time. Each sequence of three adjacent segments is compared against the constraints imposed by the grammar. If at 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 (4).

Consider once more the example derivation from (2a), repeated here:

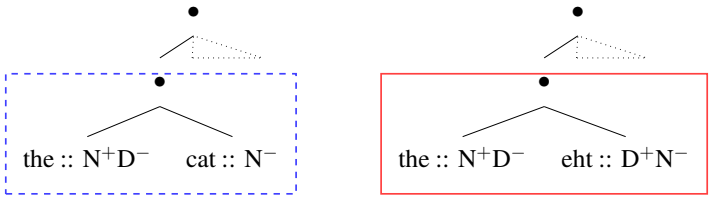
(2a′)



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 (4) given such an upper limit.

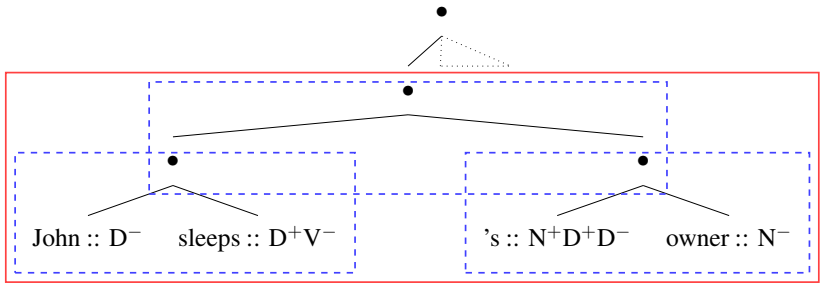
In order to determine that the merger of *the* and *cat* is licit, a window of size 2 would actually suffice. Within this window, we see that the Merge node is uniquely hosted, that full selection is satisfied for *the*, and that *cat* has a matching category feature. Hence a window of this size is also enough to block illicit configurations like the one we encountered in (5a).

(7)



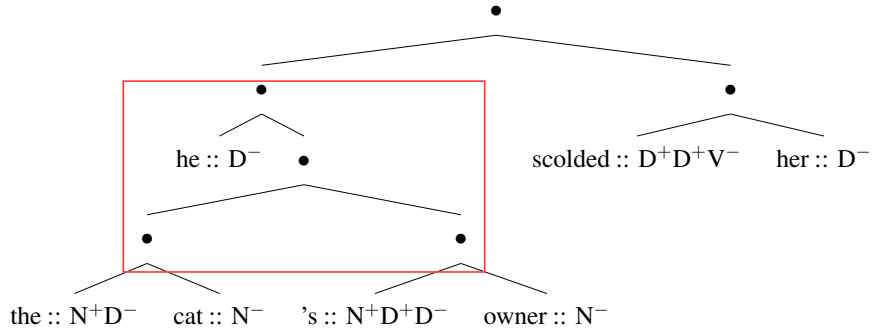
But a window of size 2 is too small for some configurations. Verifying that all Merge requirements are met with respect to *'s* demands an increase of the search window size from 2 to 3. This is illustrated in (8), where no window of size 2 can detect that the second argument of the possessive marker 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.

(8)



A size 3 window will also notice the unhosted Merge node from (5b), as is shown below.

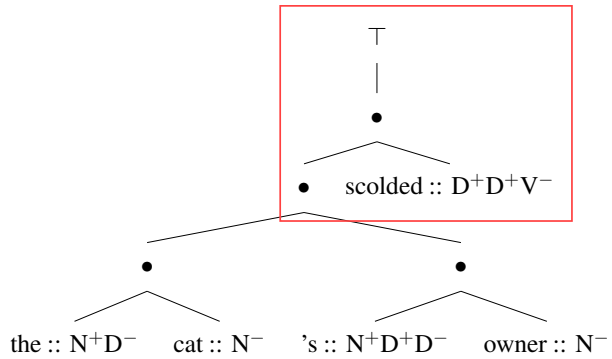
(9)



Since no LI has more than two selector features in this example MG, 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 H of the highest Merge node, provided H exists. The search window in the tree above does not contain such a H, and the whole derivation tree is thus inferred to be illicit.

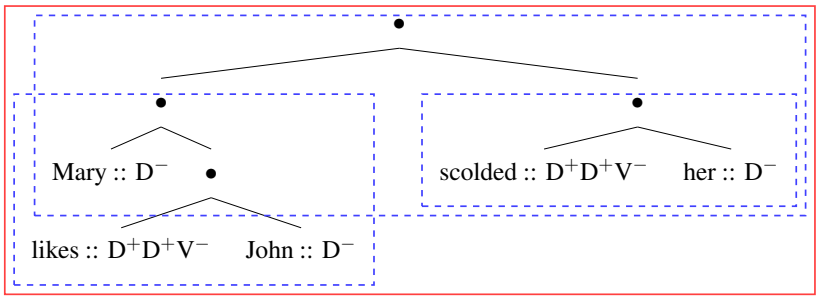
In addition, a size 3 window will recognize all full selection violations. Full selection requires an LI l with exactly n selector features to have an ancestor chain of length n. This condition is violated only if the length of the full ancestor chain of l does not exceed $n-1$. In our example, where 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 encode 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.

(10)



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 (8) 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*.

(11)



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, the following holds for every MG: if the grammar's only operation is Merge, then 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.

2.4 Non-Merge conditions on well-formedness

We have seen that Merge can be formalized as a collection of SL conditions on derivation trees. But even in a grammar whose only operation is Merge it is not guaranteed that a derivation tree that satisfies all conditions on Merge is well-formed. From a linguistic perspective this is not surprising, after all syntax contains many modules such as binding theory and case theory that are not naturally subsumed by Merge (but see Graf 2017a for a formal study of how these conditions can be enforced just with Merge). Let us put aside these phenomena and consider only MGs as they are mathematically defined in Stabler (1997). For these grammars, it still holds that well-formedness involves more than satisfying all constraints on Merge: every LI l must have its category feature checked unless l has the category feature C and the D-root of l is the root of the derivation tree. However, these additional conditions also fall within SL.

Note first that the Merge constraints in (4) already preclude that both of the following hold of any arbitrary LI l: I) the category feature of l has not been checked, or II) l is not the head of the whole tree (more precisely, the D-root of l is not the root of the derivation tree). Suppose l is not the head of the whole tree. Then the D-root of l must have a Merge node m as its mother. By the single head condition, m is hosted by some selector feature X^+ on some LI l_1 distinct from l. By the match condition, exactly one of the daughters of m must be the D-root of an LI with category feature X^-. Since Merge is binary, this can only be l. As every LI in an MG has exactly one category feature, this implies that all category features of l are checked at some point of the derivation. This establishes that as long as LIs cannot have multiple category features and Merge is binary, all LIs except the highest one are guaranteed to have their category feature checked.

The restriction to exactly one category feature and binary Merge are each SL.

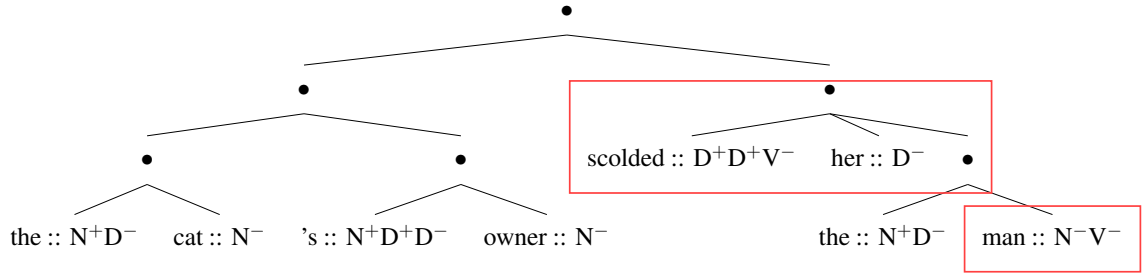
(12) **General conditions on derivation trees**

a. *Unique category*
Every LI has exactly one category feature.

b. *Binarity of Merge*
Every Merge node has at most two daughter nodes.

The unique category condition is SL-1 because it suffices to look at individual nodes. If any one of them is an LI with either 0 or at least 2 category features, the whole derivation tree is rejected. The binarity of Merge holds between a node and its daughters and thus is SL-2. Note that it is enough to only block the presence of more than two daughters because the conditions single head and match already imply that every Merge node has at least two daughters.

(13) **Violation of unique category and binarity of Merge**



This still leaves the case when l is the head of the tree. In this case, the selector feature of l does not need to be checked, but instead it must be C^-. This constraint ensures that only full CPs can be built by the grammar (in practice, one has to distinguish between matrix-CPs and embedded-CPs and only allow the former to remain unselected, but this is immaterial for the current discussion).

(14) **Final category**
If the D-root of LI l is the root of the derivation tree, then the category feature of l must be C^-.

The final category condition is SL-$(k+2)$, where k is the maximum number of positive features on an LI. So it would be SL-4 for the example grammar from §2.3. This bound is obtained as follows: It suffices to ensure that if the search window contains the special marker $\top$ (which signals that its daughter is the root of the derivation tree) then its daughter d is the D-root of an LI l with feature C^-. Every window of size $k+2$ covers $\top$, d, and l, so final category violations will not go unnoticed.

To sum up, the conditions on Merge do not fully specify the set of well-formed derivation trees in an MG with Merge but not Move. In addition, one also has to enforce the conditions unique category, binarity of Merge, and final category. This is easily done, though, because they too are SL. Even if this were not the case it would not undermine any of the claims about Merge in this paper. But it is good to be aware of these three supplementary conditions as they will make the parallels between Merge and Move surface even more clearly at the end of §4.

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 (4). 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.

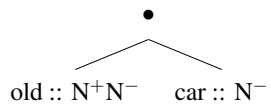
But the locality of Merge is not an intrinsic property of the operation, it depends on the grammar as a whole and can be disrupted by other operations. This is exactly what we will observe in this section. Once an adjunction operation is added to MGs, the distance between a selector and the head of its 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, §3.5). 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 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.

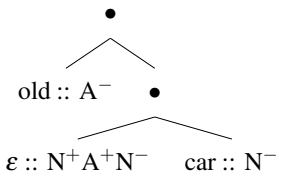
That said, there is one implementation that is even simpler than the one I will adopt. This variant reduces adjunction to a special case of selection and thus does not alter the complexity of Merge at all. For example, *old* would not be treated as an adjective that adjoins to the noun *car*, but as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

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



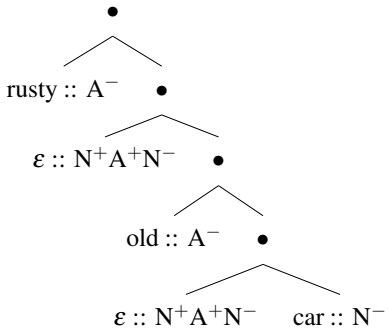
It seems strange to treat *old* as a noun, though, so a more elaborate version might instead posit an empty head $\varepsilon :: N^+A^+N^-$ which implicitly converts an adjective into an NP-adjunct (cf. the analysis of expletives in Kobele 2006:42f).

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

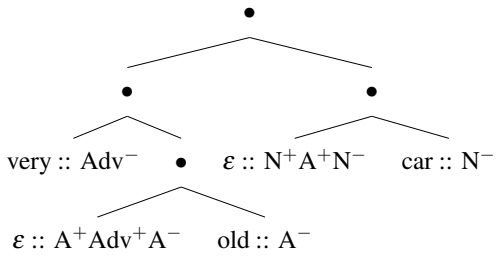


This analysis still allows for an unlimited number of adjuncts, and it is also possible to adjoin to an adjunct.

(17) a. **Multiple adjunctions**



b. **Recursive adjunction**

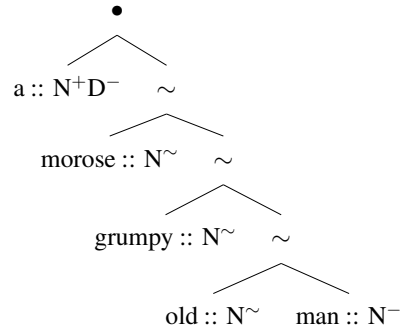


Even though this analysis is appealing in its simplicity, its conflation of arguments and adjuncts is also a detriment. Without further stipulations, this view predicts that arguments and adjuncts should behave the same. Yet there is plenty of empirical evidence to the contrary. For example, arguments 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 and Gärtner (2002) propose to model adjuncts as LIs that have an adjunction feature $X^{\sim}$ 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.

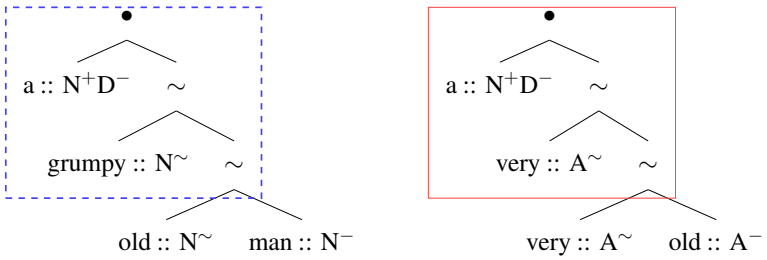
(18)



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 *a* and *man*, between which the Merge dependency holds.

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 (4) 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 $N^{\sim}$. Since an LI with feature $N^{\sim}$ must adjoin to an LI with category feature N^-, the presence of the latter can be inferred from the former.

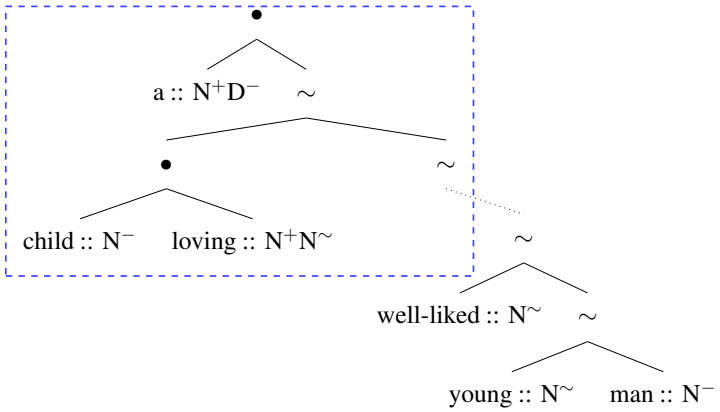
(19)



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 and 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.

(20)

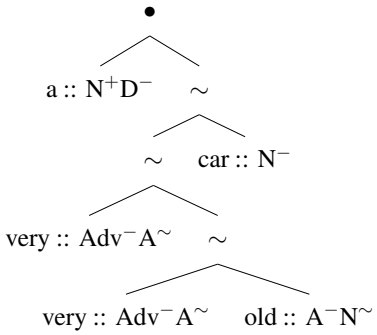


While adjuncts may take arguments, the current system makes it impossible for them to contain adjuncts of their own — adjunction can only target phrases whose head has a category feature, but category features and adjunction features are mutually exclusive. Next we will see that relaxing this condition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

(21) **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.

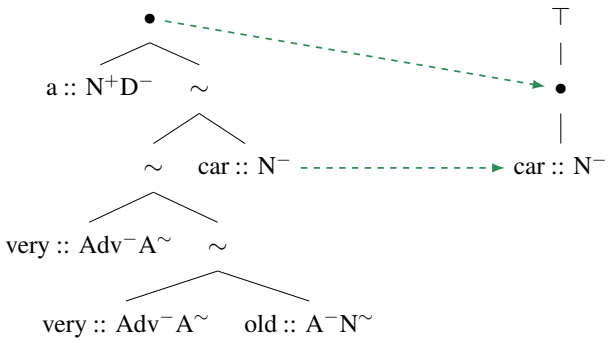
At a more abstract level, the challenge posed by recursive adjunction compared to its non-recursive counterpart is the increased difficulty in decomposing a non-local dependency into a local one. With non-recursive adjunction, the non-local Merge dependency is instead mediated by local dependencies between 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 consequently Merge is pushed out of SL.

3.4 Merge with recursive adjunction is SL over tiers

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. As I show next, 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.

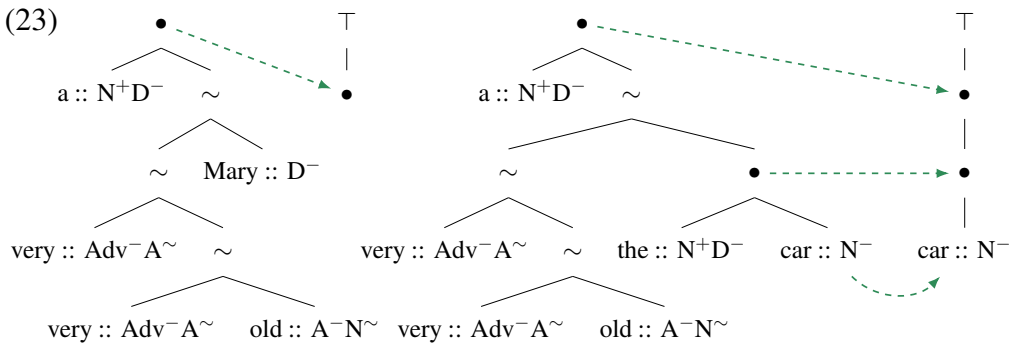
Suppose that we take the derivation tree with recursive adjunction in (21) and remove all nodes except those that either are hosted by a selector feature X^+ or carry a category feature X^- but no adjunction features. Let us call the structure produced by this removal step the *X-tier* that is projected from the derivation tree. Example (22) 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. (24)). The presence of $\top$ will also be useful in §4.

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



Now contrast (22) with the N-tiers projected from ill-formed derivations.

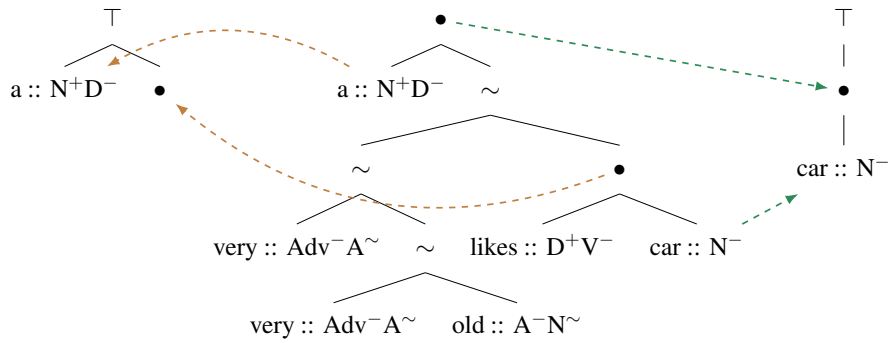
(23)



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 (4) 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.

(24)

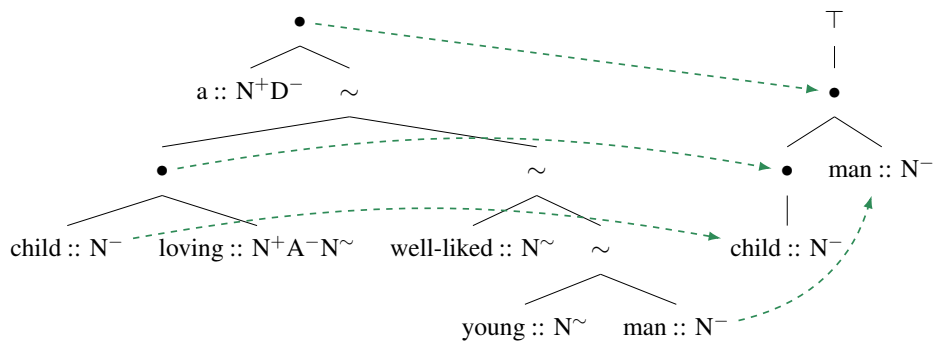


Example (24) shows that if one projects a tier for each kind of category/selector feature, then any violations of the match constraint will surface on these tiers in the form of a Merge node without an LI as a daughter.

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

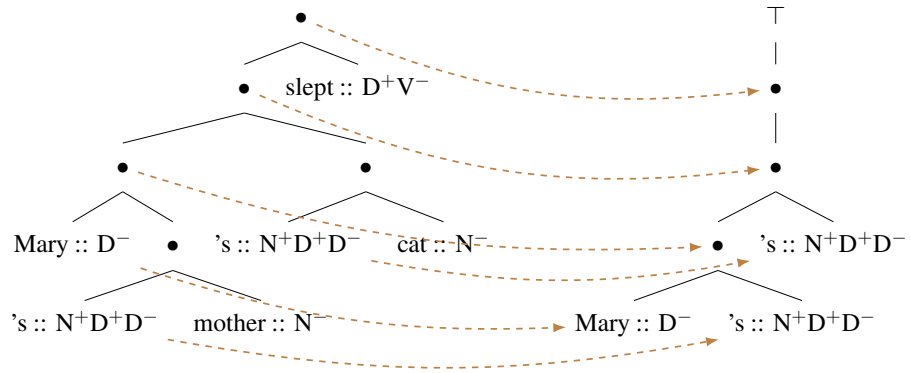
The constraint indirectly enforces the match condition on Merge as a condition on the shape of Merge tiers. It yields the desired result even in configurations where a Merge node has multiple daughters on a tier.

(26)



That said, the constraint as stated in (25) is actually too strong as it fails in cases where an LI contains both a category feature X^- and a selector feature X^+.

(27)



Here it is no longer the case that every Merge node has exactly one lexical daughter. Rather, a Merge node may be separated from an LI l by i other Merge nodes, where i is the number of X^+ features on l. Since i is bounded for every MG, this condition would still be SL, but it is more complicated than (25). Moreover, these configurations are also fairly rare across languages, with the two most notable examples being possessives and possibly serial verb constructions (cf Collins 1997:484–488). For these reasons, I will put aside such configurations for now and assume that no LI of category X selects an argument of category X.

Although the preceding discussion does not qualify as a formal proof, it should now 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. If one projects all possible tiers and each one of them is well-formed, then the derivation tree cannot contain any violations of the match constraint.

But what about the other two constraints in (4), single head and full selection? They 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. Unless one allows X'-adjunction, then, 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.

Something very similar holds for the three constraints encountered in §2.4: unique category and binarity of Merge are SL, whereas final category is SL over tiers. Recall first that unique category is SL-1 in MGs without adjunction. This means that the surroundings of a node are never taken into account. But then it does not matter at all whether the grammar allows for recursive adjunction, so unique category is SL-1 in MGs with adjunction, too. Similarly, binarity of Merge only requires every Merge node to have two daughters, which is always SL-2. Whether one of the daughters is an adjunction node is immaterial for the constraint. Therefore the SL-nature of unique category and binarity of Merge is independent of our assumptions about adjunction.

This only leaves the third constraint, final category. As stated in (14), this condition requires every LI l to be selected by some distinct LI l' unless l is the root of the derivation tree, in which case l must carry the category feature C^-. If an LI with category feature X^+ is not selected by anything, then it does not have a Merge node as its mother on the X-tier. This can be seen in (24), where a :: N^+D^- is not selected by anything and surfaces on the D-tier without a Merge node as its mother. This insight provides us with an easy way of defining final category as a constraint over tiers.

(28) **Final category (over tiers)**
An LI may be the daughter of $\top$ on an X-tier iff X = C.

This is an SL-2 constraint on tiers. So just like the match constraint, final category preserves its SL nature if it applies on tiers rather than derivation trees. Overall, then, all the established constraints on MG derivation trees — single head, full selection, match, unique category, binarity of Merge, and final category — are SL over derivation trees or SL over tiers projected from derivation trees.

3.5 Merge as an instance of TSL

The previous section has established several important facts for 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:

(29) **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 and Graf 2017). Merge requires such a structure-sensitive tier projection to determine what feature a Merge node is hosted by.

b. Each tier comes with a *licensing function* that 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. This enforces that every Merge node must have exactly one LI among its daughters. LIs are mapped to the empty set by the licensing function, which encodes that they must be leaf nodes. The special node $\top$ at the top of a tier is mapped to $\bullet^* l \bullet^*$ in C-tiers (exactly one LI is not selected by anything) and $\bullet^*$ in all other tiers.

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

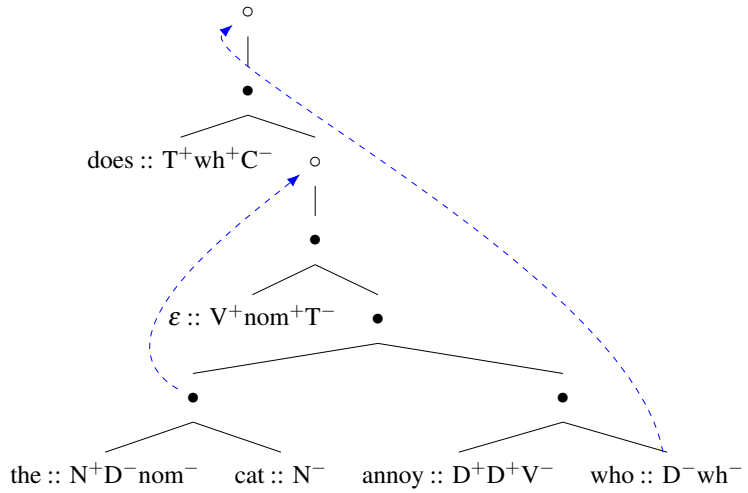
The methods I used to show that Merge is in TSL are easily employed to establish that Move is TSL, too. This was first shown in Graf and Heinz (2015), but this section presents their findings in a slightly different manner that is more in line with the TSL view of Merge. Mirroring the presentation of Merge in §2, I first give a general introduction to Move in MGs and how it can be characterized as a constraint on derivation trees (§4.1). It will quickly become clear that these constraints are SL over specific tiers, the construction of which parallels what we encountered for Merge (§4.2).

4.1 Move in MGs

The MG operation Move is modeled closely after the Minimalist notion of movement and is fully controlled by the feature calculus. 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^-. Licensor features are positive, whereas licensee features are negative. By convention, Move features are written in lower case like f^+ and f^- to distinguish them from Merge features (the only notable exception being *v*). The explicit split between Merge and Move features can be removed to bring MGs more in line with Minimalist syntax, but I will stick with it here for expository convenience. Either way it holds that an LI may occur in a well-formed derivation tree only if its feature string is of the form $\gamma c \delta$, where γ consists of 0 or more positive features (the first of which must be a selector feature), c is a category feature, and δ consists of 0 or more licensee features. So good :: $D^+f^+A^-g^-$ could possibly occur in a well-formed derivation, in stark contrast to bad :: $f^+A^-g^-D^+$. This entails that an LI undergoes up to three consecutive phases in a derivation: I) selecting arguments and attracting movers, and II) being selected by some head, and III) moving from its argument position to some higher landing site.

Although everything said so far may be familiar from Minimalist syntax, Move looks somewhat 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. The tree in (30) depicts a derivation with wh-movement and case movement that yields the surface string *who does the cat annoy*. The symbol ∘ represents instances of Move, and arrows are used as an expository device to highlight what moves where in the derived structure.

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



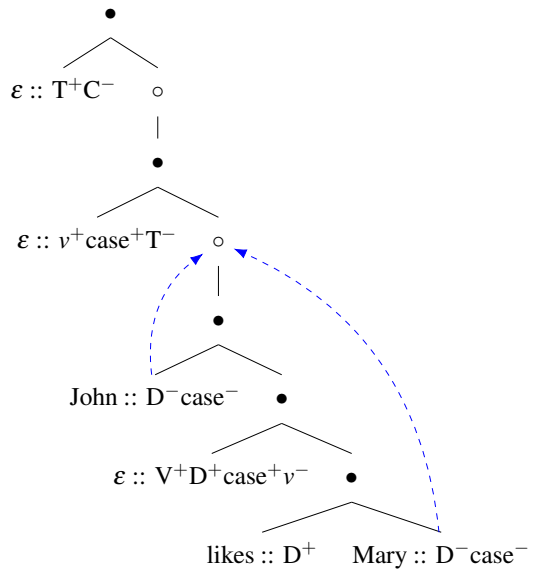
As in Minimalist syntax, checking of a licensee feature on an LI *l* triggers movement of the whole phrase headed by *l*. So in (30), the entire phrase *the cat* undergoes subject movement by virtue of being headed by *the*, which carries the relevant licensee feature nom^-. If

desired, one can also implement pied-piping by displacing an even larger subtree. Since the actual displacement is part of the process described by the derivation tree rather than the derivation tree itself, the size of the mover is irrelevant for the derivational view of Move and will be ignored for the remainder of the paper.

The reader might wonder why the arrows in (30) are merely expository devices and not part of the derivation tree itself. This is so because the arrows do not provide any information beyond what is already provided by the feature calculus. Consider an LI *l* that carries exactly one licensee feature f^-. In order to find the landing site of *l*, one simply has to look for the lowest Move node that is hosted by some f^+ and properly dominates the D-root of *l*. Let us call this Move node the *occurrence* of *l*. In (30), the higher Move node is the occurrence of *who* and the lower one the occurrence of *the*. This relation is also indirectly encoded by the arrows, but the MG feature calculus already provides all the information that is needed to match the LIs to their occurrences.

If the simple algorithm above results in configurations like (31), where two LIs receive the same occurrence, the entire derivation is discarded. The linguistically obvious alternative of picking the next higher Move nodes as the occurrence for *John* is never entertained by the grammar.

(31) **An illicit derivation with a doubly matched Move node**



The hard ban against double occurrences is known as the *Shortest Move Constraint* (SMC) in the MG literature. The SMC is fairly contentious among syntacticians as it renders Move completely deterministic and undermines standard models of syntactic locality such as Relativized Minimality (Rizzi 1990). A lengthy defense of the SMC is beyond the purview of this paper, and I will merely point out that all these proposals can in fact be implemented in a manner that is compatible with the SMC. In particular, apparent cases of competition between movers are easily recast as constraints on what features an LI may carry in specific configurations. For further details, the reader is referred to Graf (2013:111–115). The SMC

is crucial to keep the generative capacity and computational complexity of MGs in check (Salvati 2011), so I will implicitly assume it for the rest of the paper.

As stated above, the algorithm for finding an LI's occurrence is not defined for any LIs that have more than one licensee feature and thus move more than once. This is a deliberate omission because such intermediate movement is unnecessary in MGs. 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,*v*P 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 and linguistic applications); Graf and Heinz (2015) show that removing intermediate movements from the derivation tree is essential in bringing out the TSL-nature of Move.

4.2 Move as an instance of TSL

The basics of Move in MGs have now been put in place: I) Move is triggered by (positive) licensor features and (negative) licensee features, II) the SMC forbids two movers from targeting the same Move node, and III) every LI has at most one licensee feature (and hence moves at most once). As we will see next, these properties make it fairly easy to decompose Move into three constraints over derivation trees. These in turn can be converted into SL conditions on tiers.

Let us first state the relevant well-formedness conditions on Move.

(32) **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 (30), but only (32a) holds in (31). If the higher Move node were removed from (31), then (32a) would also be violated as the T-head would carry the licensor feature nom^+ without hosting any Move nodes. Any other illicit configurations are just variations of these few base cases, so that the constraints in (32) are indeed sufficient for regulating Move.

One of the most fundamental properties of Move is that there is no upper bound on the distance between an LI and its occurrence. This does not affect (32a), since every Move node is only a fixed distance away from its host (and no adjuncts can occur between the two because an LI's licensor features must have been checked before its category feature becomes available for adjunction). But (32b) and (32c), which establish dependencies between LIs and their occurrences, cannot be SL for this reason. No matter how large a

search window we pick, some occurrences will be even farther away from their LIs. Just as for Merge, though, these two constraints are SL over tiers and thus TSL.

The TSL view of Move requires a definition of the projection function and the constraints that must be satisfied by the resulting tiers. For each movement type f (e.g. nom, wh, ...) 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.

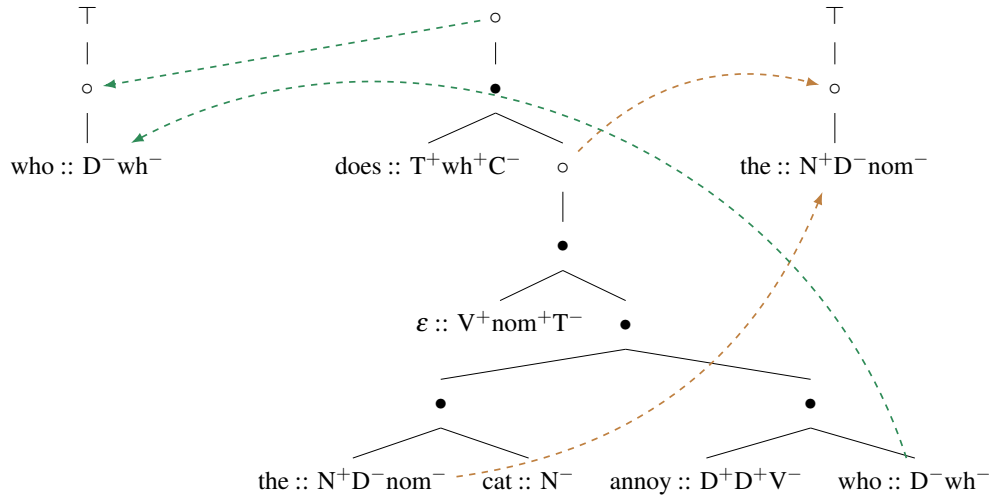
(33) **Move as constraints over tiers**

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

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

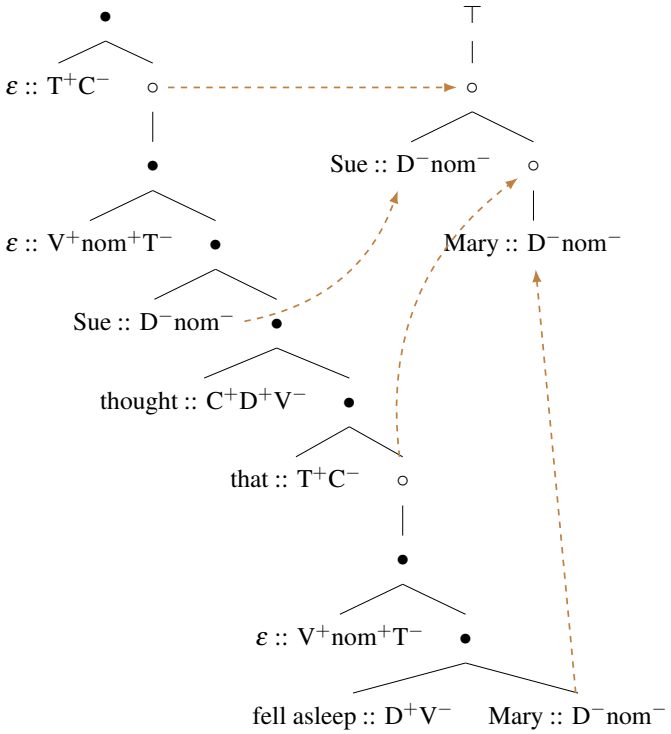
The examples below show how these conditions discriminate between well-formed and ill-formed derivation trees. The derivation tree in (34) is well-formed, and all three of its move tiers contain a Move node with exactly one lexical daughter. They thus satisfy both (33a) and (33b).

(34) **Well-formed derivation tree with tiers**



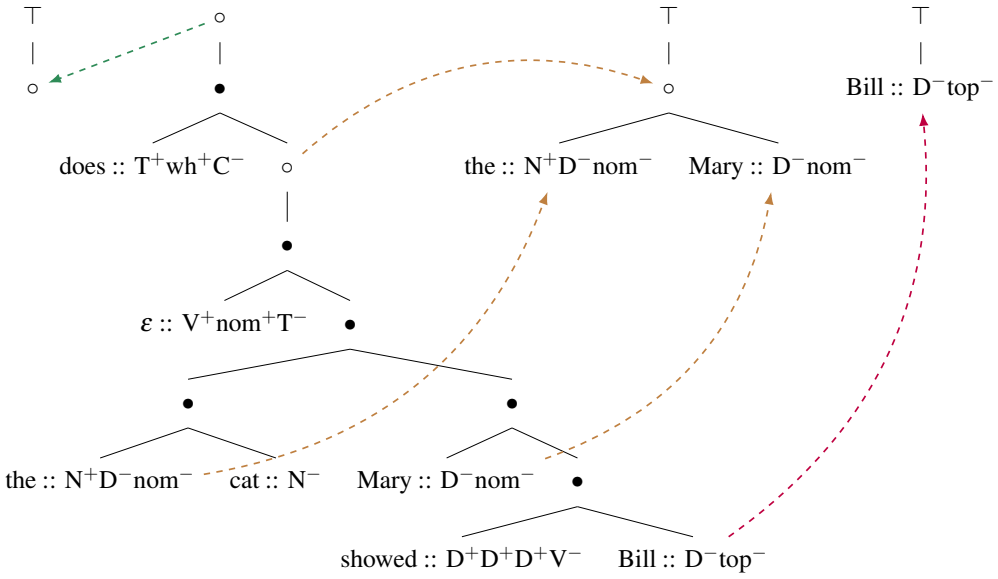
Example (35) shows that the move tiers of a well-formed derivation tree can also take on more complex shapes, but it still holds that every Move node has exactly one LI among its daughters and every LI has a Move node as its mother.

(35) **A well-formed derivation tree with a complex tier**



The ill-formed derivation tree in (36) illustrates how various violations (unchecked licensee feature, unchecked licensor feature, SMC violation) are directly reflected on the corresponding move tiers.

(36) **Ill-formed derivation tree with tiers**



Now that the constraints in (33) have been shown to correctly distinguish between well-formed and ill-formed derivation trees, they only need to be formalized in terms of TSL. From the perspective of TSL, (33a) 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 (33b) is recast in TSL-terms with the help of the assumed root node $\top$: its string of daughters must fit the pattern $\circ^*$. This, too, is familiar from Merge where all tiers except the C-tier must have $\bullet^*$ as the daughter string of $\top$. The parallelism between Merge and Move from this perspective is striking.

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

	Merge	**Move**
Projection	LI with X^- (and no $Y^\sim$)	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 $\bullet^* l \bullet^*$ for C-tier)	$\top \mapsto \circ^*$

There is still a minor wrinkle in this comparison that must be ironed out. Recall from the discussion towards the end of §3.4 that the conditions on Merge are actually more complex if an LI contains both a selector feature X^+ and the corresponding category feature X^-. The same is true for Move if an LI carries both the licensor feature f^+ and the licensee counterpart f^-. As for Merge this does not change the SL-nature of Move over tiers, but it makes the treatment more complicated. That said, the complications mirror exactly those encountered for Merge in this case, underlining the strong computational parallel between Merge and Move. Not only do Merge and Move both fall into the class TSL, they are essentially alphabetical variants with respect to how tiers are projected and regulated.

5 Discussion

This paper has argued for a specific formal view of Merge and Move in order to bring out surprising parallels between them. But the perspective is unusual, and consequently the reader may have many questions. Only the most important ones can be addressed here: the cognitive status of tiers (§5.1), the interplay of features and locality (§5.2), the connections to subregular phonology and morphology (§5.3), and the limits of TSL-syntax (§5.4).

5.1 Cognitive status of tree tiers

Derivation trees were introduced in §2.1 as a record of the computations carried out by the grammar. But the notion of tier seems at odds with this view of derivation trees. What does it mean to project a tier from a computation? When construed literally, tier projection over computations is indeed nonsensical, but as a metaphor it provides essential information about the computational mechanism.

Phonology provides a simple example to illuminate this point. Take the case of Samala, where all sibilants in a word must agree in anteriority. This is a TSL dependency, and we can think of it in a very literal fashion: we project a tier of sibilants, and on this tier we use a sliding window of size 2 to ensure that every sibilant has the same anteriority value

as the previous one. But there is a also a more abstract interpretation in terms of memory architecture (Rogers et al. 2013). Suppose that our window always has a size of 1, but we may store symbols in memory. Then a dependency is SL-n iff it suffices to memorize the previous $n-1$ symbols. Earlier symbols can be completely ignored. From this perspective, TSL is a refinement of SL where only symbols of a specific type need to be memorized — namely those that are part of the metaphorical tier. In the case of Samala, a computing device only needs to remember the last sibilant it has seen. Therefore, showing that a dependency is TSL-n does not entail that literal tier projection is involved, but rather that the dependency only requires memorizing the previous $n-1$ symbols of a specific type.

The fact that Merge and Move are both TSL thus can be interpreted as a claim about what kind of working memory has to be available to syntax. It has to furnish a separate buffer for each kind of Merge and Move feature. It also has to be sensitive to the features that are present on LIs and are checked by operations. This ensures that each "symbol" (i.e. LI or operation) is stored in the correct buffer. A realistic grammar will need many distinct buffers, but each buffer is exceedingly simple. As can be seen in (37), the only relevant factor for any given symbol on a tier is whether the daughter string contains exactly one LI. This is a binary distinction and thus can be encoded by a single bit. It is not necessary to ever memorize the full string of daughters on a tier, a single bit encodes all the necessary information. Overall, then, the cognitive architecture needed for Merge and Move is remarkably simple, and this is what TSL and its tier projection metaphor make very salient.

5.2 Interplay of features and locality

Tiers are also a metaphor for relativized locality: given the ability to ignore irrelevant material, long-distance dependencies reduce to very simple local dependencies. A system where an f-mover can move over arbitrarily many landing sites for f-movers would not be TSL. At the same time, moving over arbitrarily many landing sites for movement type g is not a problem as these do not matter for f-movement. The TSL characterization of Merge shows that relativized locality also applies for this operation. Even though the distance between selector and selectee may be arbitrarily large due to adjunction, it is local once irrelevant material is excluded. And as for movement, it is impossible to skip over potential selectees. Another insight of TSL, then, is that relativized locality is a unified principle that applies to Merge and Move alike, further deepening the parallels between the two.

The reader may object, though, that selection is always local since adjunction preserves the label of a phrase. Adjoining to a VP still produces a VP, so adjunction does not increase the distance. There are two answers to this. From a linguistic perspective, this cannot be the whole story since presumably mass nouns and count nouns both project phrases labeled NP, yet not all determiners that select count nouns can also select mass nouns. So unless one encodes all kinds of additional information in the label, adjunction still creates long-distance dependencies. But encoding all information in labels is exactly what one should not do, which takes us to the second answer.

Graf (2013, 2017a) explains how label refinement can abuse selection to enforce any

arbitrary constraint that is definable in monadic second-order logic. This is such a powerful logic that label refinement opens the door to massive overgeneration. Label refinement also destroys all subregular distinction, as every subregular class can be simplified to SL with additional diacritics. For instance, even Move becomes a local dependency if each label contains the list of active movers contained in the subtree. At this point, there are no known results as to how specific labeling systems line up with subregular complexity classes, and it is unlikely that this problem will be solved in the near future. Therefore feature coding has to be avoided as much as possible if one is interested in revealing differences in computational complexity.

This paper opted for a very minimal system of labels and features. Interior node labels are completely redundant as the distinction between Merge, Move, and Adjunction nodes can be inferred within a locally bounded context. The feature annotation of LIs only serves to factor out non-determinism, e.g. with respect to the category feature of ambiguous LIs like *export* or *water*, and to clearly identify adjuncts. This is very much in line with traditional thinking in linguistics. In addition, removing all category and selector features provably pushes Merge beyond TSL for arbitrary MGs,[1] but it might not do so for linguistically plausible MGs because of certain restrictions on subcategorization.

In sum, then, the TSL perspective has to take great care not to pack too much information into features and labels, but if done properly it clearly reveals the relativized locality of Merge and Move alike.

5.3 Subregularity across language modules

The subregular program originates from phonology, but has also been applied to morphology more recently. It is remarkable how closely the complexity results in those domains fit the ones presented here for syntax. As in syntax, the large majority of patterns in phonology and morphology fall into the classes SL and TSL.[2] This is surprising since syntax is known to be much more complex at the level of strings — whereas phonology (and morphology, to a large extent) is assumed to be regular (Johnson 1972; Kaplan and Kay 1994), syntax is known to be at least mildly context-sensitive (Joshi 1985; Shieber 1985; Michaelis and Kracht 1997; Kobele 2006). But once we recognize that syntax isn't a mechanism for generating strings and consider the complexity of the structure-building operations instead, syntax suddenly falls in line with phonology and morphology regarding subregular complexity.

It remains to be seen how far the parallels go. TSL is still a bit too weak for phonology

[1] To see why, consider an MG with only three LIs: a :: D^-, a :: D^+C^-, and a :: C^+D^-. Without category and selector features one has to use modulo counting to determine the well-formedness of derivation trees, which cannot be done with subregular means.

[2] The term TSL as originally defined for phonology in Heinz et al. (2011) is more specific than the generalization to trees in this paper. I allow tier projection to take the local context into account, and a grammar may project multiple tiers. Hence the tree version of TSL actually corresponds to the class SS-MTSL defined in De Santo and Graf (2017). However, since SS-MTSL is needed for phonology, too, the shift in terminology is innocuous.

(Graf 2017b),[3] and unbounded reduplication in morphology is not even regular. At the same time, Move is TSL only under the assumption that intermediate movement is not feature-triggered (§4.1; Graf 2018). One thus should not be too hasty to postulate a *Cognitive Parallelism Principle* such that subregular complexity is the same across all language modules, but the implications of such a principle for theoretical linguistics, cognitive science, and language biology are certainly intriguing and merit further study.

5.4 The syntactic limits of TSL

In connection with the Cognitive Parallelism Principle, it is also important to keep in mind that one cannot infer from the TSL nature of Merge and Move that all of syntax is TSL. First of all, adjunction as implemented in §3.3 is not TSL, so syntax cannot be limited to TSL. Similarly, if feature triggers for intermediate movement can be shown to be linguistically necessary (contra Graf 2018), TSL is insufficient for movement. Finally, syntax enforces a wide variety of constraints, be it Principle A and B, the Person Case Constraint, or island constraints. A commitment to TSL as an upper bound on syntactic complexity would require excluding many of them from the purview of syntax by relegating them to the interfaces.

It is worth noting, though, that positional island constraints can be expressed very easily in TSL. The adjunct island constraint, for instance, follows immediately if I) every movement tier also contains all Merge nodes that are hosted by LIs with an adjunction feature $X^{\sim}$, and II) Merge nodes on a Move tier must not have any LIs as their daughter. Any LI l that wants to move out of an adjunct will be the daughter of some Merge node, rendering the whole tier illicit. The wh-island constraint can be stated in an analogous fashion. But island constraints that require reference to multiple movements, such as the Coordinate Structure Constraint and freezing effects, are not TSL.

The enterprise of measuring the subregular complexity of syntax beyond Merge and Move is still very young (Vu 2018), so no conclusive answers are available at this point. In the same vein, little can be said about the complexity of the mappings that produce the actual output structures from derivation trees. The mathematical concepts have not been worked out yet in this area, so linguistically insightful results are still years away.

6 Conclusion

Graf (2012) argued that Merge and Move are 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).

[3] This observation holds even if one takes TSL to refer to SS-MTSL.

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, Alëna, Thomas Graf, and Sedigheh 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. URL `https://www.aclweb.org/anthology/W/W16/W16-2019.pdf`.

Chandlee, Jane. 2014. *Strictly local phonological processes*. Doctoral Dissertation, University of Delaware. URL `http://udspace.udel.edu/handle/19716/13374`.

Chandlee, Jane, and Jeffrey Heinz. 2018. Strict locality and phonological maps. *Linguistic Inquiry* 49:23–60.

Chomsky, Noam. 1986. *Knowledge of language: Its nature, origin, and use*. New York: Praeger.

Chomsky, Noam. 1995. Bare phrase structure. In *Government and binding theory and the Minimalist program*, ed. Gert Webelhuth, 383–440. Oxford: Blackwell.

Chomsky, Noam. 2004. Beyond explanatory adequacy. In *Structures and beyond: The cartography of syntactic structures volume 3*, ed. Adriana Belletti, 104–131. Oxford: Oxford University Press.

Collins, Chris. 1997. Argument sharing in serial verb constructions. *Linguistic Inquiry* 28:461–497.

De Santo, Aniello, and Thomas Graf. 2017. Structure sensitive tier projection: Applications and formal properties. Ms., Stony Brook University.

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

Frey, Werner, and Hans-Martin Gärtner. 2002. On the treatment of scrambling and adjunction in Minimalist grammars. In *Proceedings of the Conference on Formal Grammar*, ed. Gerhard Jäger, Paola Monachesi, Gerald Penn, and Shuly Wintner, 41–52.

Graf, Thomas. 2012. Locality and the complexity of Minimalist derivation tree languages. In *Formal Grammar 2010/2011*, ed. Philippe de Groot and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 208–227. Heidelberg: Springer. URL http://dx.doi.org/10.1007/978-3-642-32024-8_14.

Graf, Thomas. 2013. *Local and transderivational constraints in syntax and semantics*. Doctoral Dissertation, UCLA. URL http://thomasgraf.net/doc/papers/Graf13Thesis.pdf.

Graf, Thomas. 2014. Models of adjunction in Minimalist grammars. In *Formal Grammar 2014*, ed. Glynn Morrill, Reinhard Muskens, Rainer Osswald, and Frank Richter, volume 8612 of *Lecture Notes in Computer Science*, 52–68. Heidelberg: Springer. URL https://doi.org/10.1007/978-3-662-44121-3_4.

Graf, Thomas. 2017a. A computational guide to the dichotomy of features and constraints. *Glossa* 2:1–36. URL https://dx.doi.org/10.5334/gjgl.212.

Graf, Thomas. 2017b. The power of locality domains in phonology. *Phonology* 34:385–405. URL https://dx.doi.org/10.1017/S0952675717000197.

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

Graf, Thomas, Alëna Aksënova, and Aniello 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. Annie Foret, Glyn Morrill, Reinhard Muskens, Rainer Osswald, and Sylvain Pogodalla, 200–215. Berlin, Heidelberg: Springer. URL https://doi.org/10.1007/978-3-662-53042-9_12.

Graf, Thomas, and Jeffrey 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, Jeffrey. 2009. On the role of locality in learning stress patterns. *Phonology* 26:303–351. URL https://doi.org/10.1017/S0952675709990145.

Heinz, Jeffrey. 2015. The computational nature of phonological generalizations. URL http://www.socsci.uci.edu/~lpearl/colareadinggroup/readings/Heinz2015BC_Typology.pdf, ms., University of Delaware.

Heinz, Jeffrey, and William Idsardi. 2013. What complexity differences reveal about domains in language. *Topics in Cognitive Science* 5:111–131.

Heinz, Jeffrey, Chetan Rawal, and Herbert 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. URL http://www.aclweb.org/anthology/P11-2011.

Jackendoff, Ray. 1977. *X-bar syntax: A study of phrase structure*. Cambridge, MA: MIT Press.

Jardine, Adam. 2016. Computationally, tone is different. *Phonology* 33:247–283. URL https://doi.org/10.1017/S0952675716000129.

Johnson, C. Douglas. 1972. *Formal aspects of phonological description*. The Hague: Mouton.

Joshi, Aravind. 1985. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In *Natural language parsing*, ed. David Dowty, Lauri Karttunen, and Arnold Zwicky, 206–250. Cambridge: Cambridge University Press.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378. URL http://www.aclweb.org/anthology/J94-3001.pdf.

Kobele, Gregory M. 2006. *Generating copies: An investigation into structural identity in language and grammar*. Doctoral Dissertation, UCLA. URL http://home.uchicago.edu/~gkobele/files/Kobele06GeneratingCopies.pdf.

Michaelis, Jens, and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Artifical Intelligence*, 329–345. Springer. URL http://dx.doi.org/10.1007/BFb0052165.

Richards, Norvin. 2016. *Contiguity theory*. Cambridge, MA: MIT Press.

Rizzi, Luigi. 1990. *Relativized minimality*. Cambridge, MA: MIT Press.

Rogers, James, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal Grammar*, ed. Glyn Morrill and Mark-Jan Nederhof, volume 8036 of *Lecture Notes in Computer Science*, 90–108. Springer. URL http://dx.doi.org/10.1007/978-3-642-39998-5_6.

Salvati, Sylvain. 2011. Minimalist grammars in the light of logic. In *Logic and grammar — essays dedicated to Alain Lecomte on the occasion of his 60th birthday*, ed. Sylvain Pogodalla, Myriam Quatrini, and Christian Retoré, number 6700 in Lecture Notes in Computer Science, 81–117. Berlin: Springer.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8:333–345. URL http://dx.doi.org/10.1007/BF00630917.

Stabler, Edward P. 1997. Derivational Minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer. URL https://doi.org/10.1007/BFb0052152.

Stabler, Edward P. 2011. Computational perspectives on Minimalism. In *Oxford handbook of linguistic Minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.

Steedman, Mark. 2001. *The syntactic process*. Cambridge, MA: MIT Press.

Vu, Mai Ha. 2018. Towards a formal description of NPI-licensing patterns. In *Proceedings of the Society for Computation in Linguistics*, volume 1, 154–163.

Changelog

V3

- fixed some typos and minor mistakes

V2

- split introduction into abstract + introduction
- detailed example for Merge feature checking
- define host in terms of positive and negative
- rename uniqueness to single head
- new subsection 2.4 on what category features must be checked
- split 3.4 into 3.4 and 3.5
- more detailed discussion of final category condition
- greatly expanded discussion of Move
- new Section 5 discussing various big-picture issues

V1

original version; to appear in Proceedings of CLS 2017