Multiple Wh-Movement is not Special: The Subregular Complexity of Persistent Features in Minimalist Grammars

Thomas Graf
Department of Linguistics
Stony Brook University
Stony Brook, NY 11794, USA
mail@thomasgraf.net

Kalina Kostyszyn
Department of Linguistics
Stony Brook University
Stony Brook, NY 11794, USA
kalina.kostyszyn@stonybrook.edu

Abstract

Minimalist grammars have been criticized for their inability to analyze successive cyclic movement and multiple wh-movement in a manner that is faithful to the Minimalist literature. Persistent features have been proposed in the literature as a potential remedy (Stabler, 2011; Laszakovits, 2018). We show that not all persistent features are alike. The persistent features involved in multiple wh-movement do not increase subregular complexity, making this phenomenon appear very natural from the perspective of MGs. The persistent features in successive-cyclic movement, on the other hand, change the subregular nature of movement, favoring an alternative treatment along the lines of Kobele (2006).

1 Introduction

Minimalist syntax assumes that movement is feature-triggered, but in many cases a single feature may trigger multiple movement steps. For example, a single *wh*-feature on a phrase may cause it to undergo a number of successive-cyclic movement steps to the matrix clause, as in (1a). In (1b), a single subject undergoes a number of raising steps, which are sometimes analyzed as being driven by a single feature on the subject.

(1) a. [Which car]$_i$ did Mary say [t_i that John thinks that [t_i that Sue mentioned [t_i that Bill bought t_i]]].

b. [The car]$_i$ seems to Mary [t_i to appear [t_i to have been expected [t_i to win the race]]].

Alternatively, a single feature may cause multiple independent phrases to move to the same position, as in the Serbo-Croatian example of multiple wh-movement below (cf. Bošković, 2002, p. 353).

(2) [Ko$_i$ koga$_j$ [t_i voli t_j]]?
who whom loves

Minimalist grammars (MGs) are a formalization of Minimalist syntax that aims for a high degree of faithfulness so that common Minimalist analyses can be easily recast in terms of MGs. But MGs are built on the assumption that each movement step is triggered by a pair of matching movement features — one on the mover, one on the landing site. By default, a single feature cannot trigger multiple movement steps, which is at odds with the very common analyses sketched above.

Multiple solutions have been provided in the literature. Kobele (2006, p. 84, 148) suggests that cases like (1a) and (1b) may involve only a single feature at the target site, with the intermediate traces inserted as part of a single movement step from the base position to the surface position in the matrix clause. This way, MGs can retain their original feature calculus while generating exactly the kind of phrase structure trees that syntacticians posit in their analyses (and which may be used in post-syntactic steps, e.g. semantic interpretation in the vein of Heim and Kratzer 1998). Stabler (2011) and Laszakovits (2018), on the other hand, extend MGs with *persistent features* which can participate in multiple movement steps. The *persistent licensee features* of Stabler (2011) handle the phenomena in (1), whereas the *persistent licensor features* of Laszakovits (2018) can be used to model multiple wh-movement (although that was not the original intent). At this point, little is known about the formal impact of persistent features except that persistent licensee features do not increase weak generative capacity (Stabler, 2011).

Recent developments make this a more urgent issue. MGs have been studied from the perspective of subregular complexity (Graf, 2018; Graf and De Santo, 2019), which is more fine-grained than the measures previously used in the MG literature. As a result, even minor changes in the formalism can greatly impact subregular complex-

Proceedings of the Society for Computation in Linguistics (SCiL) 2021, pages 275-285.
Held on-line February 14-19, 2021

ity, which may cause the reader to wonder why anybody would want to use such a finicky notion of complexity.

The subregular approach to syntax has three major advantages: First, it provides novel ways of limiting overgeneration in a principled fashion and thus furnishes new explanations of typological gaps and linguistic universals, for instance why no syntactic phenomenon seems to involve modulo counting (Graf, 2020). Second, it also reveals surprising parallels between syntax on the one side and phonology and morphology on the other. Although the weak generative capacity of phonology and morphology (Kaplan and Kay, 1994) is much less than that of syntax (Huybregts, 1984; Shieber, 1985; Radzinski, 1991; Michaelis and Kracht, 1997; Kobele, 2006), the complexity of syntactic dependencies over trees is comparable to that of phonological and morphological dependencies over strings. The three language modules apparently share a lot of computational machinery, and this makes it easier to transfer empirical and theoretical insights between them. Finally, there is a number of efficient learning algorithms for subregular string languages (Heinz et al., 2012; Jardine and McMullin, 2017; McMullin et al., 2019), and these algorithms are easy to lift to tree languages; if one assumes that the input to the child learner includes some tree structure (e.g. semantic head-argument relations), one of the subregular learning algorithms could perhaps be extended into a new learning algorithm for MGs and hence a form of Minimalist syntax. Overall, then, the subregular approach holds a lot of promise, and by studying the subregular complexity of syntactic proposals, we get a deeper understanding of the empirical viability of this approach. Since persistent features are so common in Minimalist analyses, it is important to understand their impact on subregular complexity, whether all types of persistent features are the same from this perspective, and what this entails for theoretical syntax.

In this paper, we show that the subregular view of MGs reveals a split between persistent licensor features on the one hand and persistent licensee features on the other. Persistent licensor features are a simple modification of MGs that has no impact on their subregular complexity. Persistent licensee features, on the other hand, are both more complicated and more complex. When MGs are defined in first-order logic, adding persistent licensor features is a matter of changing a single quantifier, whereas persistent licensee features require much more elaborate modifications. In addition, MGs with persistent licensee features lose the connection between movement and TSL-2 tree grammars, which form the subregular backbone of standard MGs. All of this makes persistent licensee features more complex than persistent licensor features.

We argue that these findings support two linguistic claims: First, the treatment of successive-cyclic movement in Kobele (2006) as an epiphenomenon without dedicated feature triggers is preferable from a subregular perspective. Additional work should be done on whether existing syntactic analyses can be easily revised to be compatible with this approach. Second, the existence of multiple wh-movement is unsurprising because the computational machinery that is needed for standard movement in MGs already furnishes all the power that is needed for persistent licensor features, which can give rise to this phenomenon.

On the way towards this result, we establish several formal milestones that we hope will be useful for future work. We begin with a definition of MGs in first-order logic (Sec. 2.1, Fig. 2 and 3). In contrast to earlier definitions (Graf, 2012), ours builds on MG dependency trees, which have seen increased usage in work on subregular complexity. Section 2.2 then enhances the basic version of MGs with persistent features, which is much easier for persistent licensor features than for persistent licensee features. After that, we switch from first-order logic to the much more limited class of TSL-2 tree languages (Sec. 3). Even though several publications have already used the concept of TSL tree languages (Vu, 2018; Vu et al., 2019), this paper is the first one to include the full formal definition (Sec. 3.1, Def. 1). It is already known that MGs in single movement normal form can be described as the intersection of multiple TSL-2 tree languages, and we show that this result holds also if one adds persistent licensor features to such MGs (Sec. 3.2). However, the behavior of persistent licensee features cannot be handled correctly with TSL-2.

Due to space constraints, we unfortunately have to presuppose familiarity with MGs and subregular complexity. The reader may consult Stabler (2011) for additional details on MGs, Graf (2018) for an intuitive discussion of TSL, and Cornell and Rogers (1998) for a primer on model-theoretic syntax.

2 MGs and first-order logic

2.1 Standard MGs

In MGs, all syntactic work is done by the feature calculus. Let $Merge$ and $Move$ be finite, disjoint sets of Merge feature names and Move feature names, respectively. In addition, the set $Pol := \{+, -\}$ contains two opposite *polarities*. Then $Feat := (Merge \cup Move) \times Pol$ is the set of all possible features. We adopt the following terminology and notation for features:

Type	Pol.	Name	Notation
$Merge$	$+$	Selector feature	F^{+}
$Merge$	$-$	Category feature	F^{-}
$Move$	$+$	Licensor feature	f^{+}
$Move$	$-$	Licensee feature	f^{-}

Given a finite set Σ of phonetic exponents, a *Minimalist grammar* (MG) over Σ is a finite subset G of $\Sigma \times Feat^{+}$. Each member of G is called a *lexical item*, and we write $\sigma :: \mathrm{f}_1 \cdots \mathrm{f}_\mathrm{n}$ instead of $\langle \sigma, f_1 \cdots f_n \rangle$. Each lexical item has a feature string where

1. all negative features follow all positive features,
2. there is exactly one category feature, and this feature precedes all other negative features,
3. if there are any positive features, the first one is a selector feature.

These are corollaries of the MG feature calculus.

In standard MGs, lexical items are combined into a phrase structure tree via repeated application of the feature checking operations *Merge* and *Move* until the only remaining feature is C^{-} on the head of the assembled phrase structure tree (Stabler, 1997; Stabler and Keenan, 2003). Rather than defining these mechanisms directly, we follow the two-step approach to MGs instead (Morawietz, 2003; Mönnich, 2006; Kobele et al., 2007; Graf, 2012). From this perspective, each MG is factored into two components. One is a regular language of abstract trees, the other is a *spell-out* mapping that translates the abstract tree into the desired output structure (e.g. a string, a phrase structure tree, or a logical form). From this perspective, syntax is about separating well-formed abstract trees from ill-formed ones, and each Minimalist grammar can be equated with its set of well-formed abstract trees.

For MGs, these abstract trees are often equated with MG derivation trees (Fig. 1, left), but we adopt a specific format of dependency trees (Fig. 1, right) as this will simplify the discussion of subregular complexity in Sec. 3. In the dependency tree format, the daughters of a node are its arguments, with the n-th argument corresponding to the daughter with $n - 1$ right siblings. For the reader's convenience, Fig. 1 also contains dashed movement arrows to indicate how each licensee feature is matched up with the closest available licensor feature, which, following Graf 2012, we call its *occurrence*. This illustrates how the abstract trees provide all the necessary information to obtain the corresponding string or phrase structure tree. The properties of the MG feature calculus are such that the set of abstract trees can be defined in first-order logic and hence is (sub)regular — this holds for derivation trees as well as dependency trees.

The first-order definition of an MG's set of well-formed dependency trees is given in Fig. 2 and Fig. 3. It uses first-order logic with the standard quantifiers and boolean connectives, as well as the *proper dominance* relation $\lhd^{+}$ and the *sibling precedence* relation $\prec$: $x \lhd^{+} y$ iff x properly dominates y, and $x \prec y$ iff x is a left sibling of y. In addition, we treat each lexical item $\sigma :: \mathrm{f}_1 \cdots \mathrm{f}_\mathrm{n}$ as a unary predicate that acts as the label of a node in the tree. This allows us to define ancillary predicates like $\mathrm{sel}_{i@j}(x)$, which are just finite disjunctions of labels that meet the relevant condition. In combination with a standard axiomatization of finite trees (e.g. Backofen et al. 1995), these first-order formulas yield exactly the set of well-formed dependency trees for any given MG.

To the best of our knowledge, this is the first time a first-order definition of MG dependency trees is presented in the literature. The central intuition is that MGs involve dependencies between features. In the case of Merge, this is easy. First, the number of daughters of l must match exactly the number of selector features on l. Furthermore, the category feature of the i-th argument of a lexical item l must match the i-th selector feature of l. Note that the i-th argument is the i-th daughter from the right, and that i-th selector feature of l is not necessarily the same as the i-th feature of l because licensor and selector features can be interspersed (hence the distinction between f^{p}_{i} and $\mathrm{f}^{p}_{@m}$ in Fig. 2). Once this distinction is accounted for, checking Merge reduces to checking a head's selector features against

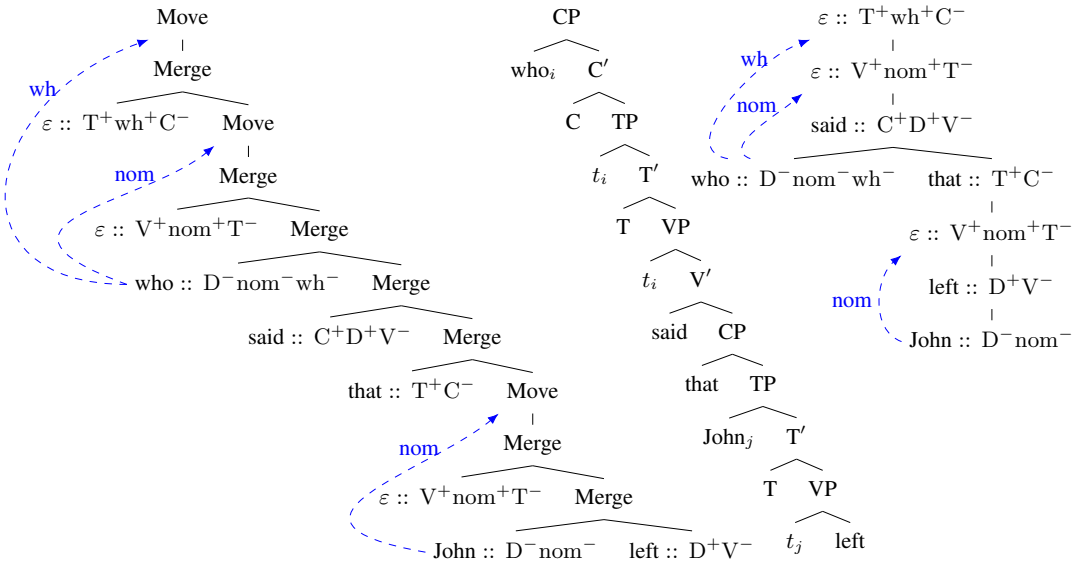


Figure 1: MG derivation tree, phrase structure tree, and MG dependency tree, for *Who said that John left* (under a simplified analysis)

the category features of its arguments (Fig. 3).

Move is more involved as it requires a switch from dominance between nodes to dominance between features. For each licensee feature f^- on a lexical item l, a matching licensor feature f^+ has to be available on another lexical item l' that properly dominates l. But not every licensor feature on l' may be available for checking. Suppose that l' has the feature string $\mathrm{A}^+\mathrm{f}^+\mathrm{B}^+\mathrm{g}^+\mathrm{C}^-$, and that l is part of the argument that is selected via B^+. In this case f^+ is not available for l because the MG feature calculus requires all features preceding B^+ to have already been checked before selection via B^+ takes place. Among the licensor features of l', only those after B^+ are available. Hence it is not enough that l' properly dominates l, we have to extend dominance to features such that we can tell which features of l' properly dominate the relevant licensee feature of l. This is the job of $\blacktriangleleft^+$, where $x \blacktriangleleft^+_{m,n} y$ means that the m-th feature of x properly dominates the n-th feature of y. With this notion of feature-dominance, we can define the matching licensor features in a recursive fashion. The *zero occurrence* of lexical item l is the matching selector feature on the mother of l. The predicate $\mathrm{occ}_{0@m}(x, l)$ is read as "the m-th feature on x is the zero occurrence of l". If l has any licensee features, we then try to find a match for its first licensee feature. This is the closest licensor feature that properly dominates the zero occurrence of l, and this feature is the *first occurrence of* l. We continue in the same fashion for each licensee feature of l, always looking for the closest matching licensor feature that properly dominates the previous occurrence. The occurrences of l thus are the specific licensor features that check licensee features of l.

Standard MGs then impose two constraints on movement. Every licensee feature must have a matching occurrence (Move in Fig. 3), and every licensor feature is an occurrence for exactly one lexical item (SMC in Fig. 3). This creates a one-to-one matching requirement where every licensee feature checks exactly one licensor feature, and the other way round. Relaxing the one-to-one matching requirement allows for persistent licensee and/or licensor features. In the next section, we will see that such a modification does not change the first-order nature of the dependency trees. Section 3 then shows that more limited, subregular notions of complexity do reveal a difference in complexity between the standard feature calculus and the relaxed version.

2.2 Adding persistent features to MGs

MGs' one-to-one matching between licensee and licensor features can be relaxed in two directions: a single licensee feature could check multiple licensor features, and a single licensor feature could check multiple licensee features. The former case of *persistent licensee features* was first defined in

$$x \lhd y \stackrel{\text{def}}{=} x \lhd^+ y \wedge \neg\exists z[x \lhd^+ z \wedge z \lhd^+ y] \tag{1}$$

$$\exists!^n x[\phi(x)] \stackrel{\text{def}}{=} \begin{cases} \neg\exists x[\phi(x)] & \text{if } n = 0 \\ \exists x_1, \ldots x_n \Big[\bigwedge_{1 \leq i \neq j \leq n} \neg(x_i \approx x_j) \wedge & \text{otherwise} \\ \phi(x_i) \wedge \forall y[\phi(y) \rightarrow \bigvee_{1 \leq i \leq n} y \approx x_i] \Big] & \end{cases} \tag{2}$$

$$\mathrm{f}^p_i(x) \stackrel{\text{def}}{=} \mathrm{f}^p \text{ is the } i\text{-th Move feature of polarity } p \tag{3}$$

$$\mathrm{F}^p_i(x) \stackrel{\text{def}}{=} \mathrm{F}^p \text{ is the } i\text{-th Merge feature of polarity } p \tag{4}$$

$$\mathrm{f}^p_{@j}(x) \stackrel{\text{def}}{=} \mathrm{f}^p \text{ is the } j\text{-th feature of } x \tag{5}$$

$$\mathrm{f}^p_{i@j}(x) \stackrel{\text{def}}{=} \mathrm{f}^p_i(x) \wedge \mathrm{f}^p_{@j}(x) \tag{6}$$

$$\mathrm{sel}_{i@j}(x) \stackrel{\text{def}}{=} \text{the } i\text{-th selector feature of } x \text{ is its } j\text{-th feature} \tag{7}$$

$$\mathrm{arg}_n(x, y) \stackrel{\text{def}}{=} x \lhd y \wedge \exists!^{n-1} z[y \prec z] \tag{8}$$

$$\mathrm{occ}_{0@m}(x, y) \stackrel{\text{def}}{=} \bigvee_{1 \leq i \leq m} \big(\mathrm{arg}_i(x, y) \wedge \mathrm{sel}_{i@m}(x)\big) \tag{9}$$

$$\phi(x, y, m) \stackrel{\text{def}}{=} \exists z \Big[(z \approx y \vee z \lhd^+ y) \wedge \Big(\bigvee_{1 \leq i \leq m \leq \Delta} \mathrm{occ}_{0@i}(x, z) \Big) \Big] \tag{10}$$

$$x \blacktriangleleft^+_{m,n} y \stackrel{\text{def}}{=} \begin{cases} x \approx y \vee \phi(x, y, m) & \text{if } m > n \\ \neg(x \approx y) \wedge \phi(x, y, m) & \text{if } m = n \\ \phi(x, y, m) & \text{otherwise} \end{cases} \tag{11}$$

$$\begin{aligned} \mathrm{occ}_{n@m}(x, y) \stackrel{\text{def}}{=} \bigvee_{f \in Move} \Big(& \mathrm{f}^-_n(y) \wedge \mathrm{f}^+_{@m}(x) \wedge \\ & \exists z \Big[\bigvee_{1 \leq i \leq \Delta} \Big(\mathrm{occ}_{n-1@i}(z, y) \wedge x \blacktriangleleft^+_{m,i} z \wedge \\ & \neg\exists z' \big[\bigvee_{1 \leq j \leq \Delta} \big(\mathrm{f}^+_{@j}(z') \wedge x \blacktriangleleft^+_{m,j} z' \wedge z' \blacktriangleleft^+_{j,i} z \big) \big] \Big) \Big] \Big) \end{aligned} \tag{12}$$

Figure 2: Ancillary first-order predicates for defining MG dependency tree languages; Δ is a suitable finite cutoff point such as the length of the grammar's longest feature string (in some cases, a lower threshold may suffice).

$$\forall x \big[\neg\exists y[y \lhd x] \rightarrow \mathrm{C}^-(x) \big] \tag{Final}$$

$$\forall x \Big[\bigwedge_{\substack{F \in Merge \\ 1 \leq i < \Delta}} \Big(\mathrm{F}^+_i(x) \leftrightarrow \exists y \big[\mathrm{F}^-(y) \wedge \mathrm{arg}_i(x, y) \big] \Big) \Big] \tag{Merge}$$

$$\forall x \Big[\bigwedge_{\substack{f \in Move \\ 1 \leq i < \Delta}} \big(\mathrm{f}^-_i(x) \rightarrow \exists y [\bigvee_{1 \leq j < \Delta} \mathrm{occ}_{i@j}(y, x)] \big) \Big] \tag{Move}$$

$$\forall x \Big[\bigwedge_{\substack{f \in Move \\ 1 \leq m < \Delta}} \big(\mathrm{f}^+_{@m}(x) \rightarrow \exists!^1 y [\bigvee_{1 \leq i < \Delta} \mathrm{occ}_{i@m}(x, y)] \big) \Big] \tag{SMC}$$

Figure 3: First-order constraints on dependency trees for standard MGs

Stabler (2011) to model successive cyclic movement, and Stabler proved that MGs with persistent features are weakly equivalent to standard MGs. *Persistent licensor features*, on the other hand, were recently proposed in Laszakovits (2018) to handle specific instances of Case licensing, but they are also implicitly assumed in some Minimalist syntax analyses of multiple wh-movement (see Gärtner and Michaelis 2010 for an MG-treatment of multiple wh-movement without persistent licensor features). Neither persistent licensee features nor persistent licensor features are lacking in syntactic applications.

The formal implementation of persistent licensor features is easier than that of persistent licensee features, so we consider the former first. The SMC in Fig. 3 states that every licensor feature is an occurrence for exactly one lexical item. If we change this to "at least one" (by replacing $\exists!^1 y$ with $\exists y$), each licensor feature becomes persistent and can serve as an occurrence for distinct lexical items. However, the feature is not persistent in the sense that it can check multiple licensee features on the same lexical item. Suppose l has the feature string $\mathrm{A}^-\mathrm{f}^-\mathrm{f}^-$, so that it has to undergo two instances of f-movement. In this case, it is not enough for the dependency tree to contain a single persistent f^+. This feature would be the first occurrence of l, but due to how we defined $\blacktriangleleft^+$ there is no other f^+ in the dependency tree that properly dominates the first occurrence of l, which means that l lacks the second occurrence. Modifying the SMC thus gives rise to a particular kind of persistent licensor feature that still behaves like a normal licensor feature with respect to any fixed lexical item, but can nonetheless serve as n occurrences for n distinct licensor features.

We can also mix persistent and non-persistent features, e.g. by defining Move features as triples drawn from $(Merge \cup Move) \times Pol \times \{1, \infty\}$, where the third component encodes whether the feature is persistent. Then the standard SMC would hold for standard licensor features, and the relaxed version for persistent licensor features. This is the system defined in Fig. 4.

In contrast to persistent licensor features, persistent licensee features require a relaxed notion of *occurrence*. A few concrete examples illustrate best how this ought to work. Suppose that the lexical item l has the feature string $\mathrm{A}^-\mathrm{f}^-_\infty\mathrm{g}^-_\infty$, where both licensee features are persistent. The licensee feature g^-_∞ on l could check every g^+ that properly dominates the second occurrence of l. The feature f^-_∞ on l is more limited, though. We do not want l to undergo any more f-movement after it has started g-moving, so the persistent f^-_∞ on l should only be allowed to check instances of f^+ that properly dominate the first occurrence of l and are properly dominated by the second occurrence of l. But this, too, must be restricted further. Suppose that f^+ is the i-th occurrence of some other lexical item l'. Unless f^+ is persistent, it must not be checked by f^- on l' and the persistent f^-_∞ on l. Instead, this occurrence of l' should also act as an upper boundary, just like the g-occurrence of l. Intuitively, then, a persistent licensee feature f^-_∞ of l can keep checking instances of f^+ after its occurrence until we reach I) the next occurrence of l, or II) an occurrence of some f^- on some other lexical item.

This system is captured in first-order logic by defining the notion of *extended occurrence* ecc such that every licensor feature must be an occurrence or an extended occurrence of some lexical item (exactly one if the licensor feature is not persistent, at least one otherwise). Even though the constraints in Fig. 4 are still fairly simple, the predicate ecc hides a very convoluted reasoning mechanism. This is indicative of the complicated nature of persistent licensee features relative to persistent licensor features, which will become even more apparent in the next section on TSL and movement.

3 Persistent features are (not) TSL

Subregular phonology has focused a lot on the class of *tier-based strictly k-local* (TSL-k) string languages (Heinz et al., 2011; De Santo and Graf, 2019; Lambert and Rogers, 2020), and while this class has also been extended to trees recently, no fully rigorous definition has been provided so far. We do so here for TSL-2 tree languages and discuss how this notion relates to movement in MGs (Sec. 3.1), after which we add persistent features into the mix (Sec. 3.2).

3.1 The TSL-nature of standard MGs

Standard MGs cannot be captured in terms of TSL, they have to be converted into *single movement normal form* (SMNF) first. An MG is in SMNF iff every lexical item has at most one licensee feature and at most one licensor feature. Graf et al. (2016) show how this is done, and that it has no effect on

$$\infty_{@m}(x) \stackrel{\text{def}}{=} \text{the } m\text{-th feature of } x \text{ is persistent} \tag{13}$$

$$\begin{aligned}\mathrm{ecc}_{n@m}(x,y) \stackrel{\text{def}}{=} \bigvee_{\substack{f \in Move \\ 1 \le i < \Delta}} \Big(& \mathrm{f}^{+}_{@m}(x) \wedge \mathrm{f}^{-}_{n@i}(y) \wedge \infty_{@i}(y) \wedge \\ & \exists z \Big[\bigvee_{1 \le j < \Delta} \Big(\mathrm{occ}_{n@j}(z,y) \wedge x \blacktriangleleft^{+}_{m,j} z \wedge \\ & \neg \exists z', y' \Big[\bigvee_{1 \le k,u < \Delta} \Big(\big(\mathrm{occ}_{n+1@u}(z',y) \vee (\mathrm{occ}_{k@u}(z',y') \wedge \mathrm{f}^{+}_{@u}(z')) \big) \wedge \\ & x \blacktriangleleft^{+}_{m,u} z' \wedge z' \blacktriangleleft^{+}_{u,j} z \Big) \Big] \Big) \Big] \Big) \end{aligned} \tag{14}$$

$$\forall x \Big[\bigwedge_{\substack{f \in Move \\ 1 \le m < \Delta}} \big(\mathrm{f}^{+}_{@m}(x) \wedge \neg \infty_{@m}(x) \rightarrow \exists !^{1} y \big[\bigvee_{1 \le i < \Delta} \mathrm{occ}_{i@m}(x,y) \vee \mathrm{ecc}_{i@m}(x,y) \big] \big) \Big]$$

(non-persistent licensor SMC)

$$\forall x \Big[\bigwedge_{\substack{f \in Move \\ 1 \le m < \Delta}} \big(\mathrm{f}^{+}_{@m}(x) \wedge \infty_{@m}(x) \rightarrow \exists y \big[\bigvee_{1 \le i < \Delta} \mathrm{occ}_{i@m}(x,y) \vee \mathrm{ecc}_{i@m}(x,y) \big] \big) \Big]$$

(persistent licensor SMC)

Figure 4: First-order constraints on dependency trees for MGs with standard and persistent features

MGs' weak or strong generative capacity (*modulo* the presence/absence of unpronounced material). The set of well-formed dependency trees of a standard MG in SMNF is *tier-based strictly local* (TSL) in a specific sense.

Definition 1 (TSL-2 over trees). Given some alphabet Σ, a *tree tier alphabet* is some fixed subset T of Σ. For every tree t over Σ and node n in t with label l, $T(n)$ is true iff $l \in T$. We define the *tier mother-of* relation $\lhd_T$ as follows:

$$\begin{aligned} x \lhd_T y \stackrel{\text{def}}{=} \; & T(x) \wedge T(y) \wedge x \lhd^{+} y \wedge \\ & \neg \exists z [T(z) \wedge x \lhd^{+} z \wedge z \lhd^{+} y] \end{aligned}$$

In addition, the *tier sibling precedence* relation $\prec_T$ is given by:

$$\begin{aligned} & x \prec_T y \stackrel{\text{def}}{=} \exists z [z \lhd_T x \wedge z \lhd_T y] \wedge \exists z, z' [\\ & (z \approx x \vee z \lhd^{+} x) \wedge (z' \approx y \vee z' \lhd^{+} y) \wedge z \prec z'] \end{aligned}$$

Then $T(t)$, *T-tier projected from t*, consists of all nodes in t with a label in T, ordered by $\lhd_T$ and $\prec_T$. In addition, if a node n has no mother on $T(t)$, then it is considered a $\lhd_T$-daughter of the tier root marker $\rtimes$. If n has no $\lhd_T$-daughter, then it is considered a $\lhd_T$-mother of the bottom marker $\ltimes$.

An *SL-2 function* S maps members of $\Sigma \cup \{\rtimes\}$ into the powerset of $(\Sigma \cup \{\ltimes\})^*$. A *tier-based strictly 2-local tree grammar over X-strings* (TSL-2[X]) is a triple $G := \langle \Sigma, T, S \rangle$ such that every set in the range of S is a string language belonging to class X. The tree language $L(G)$ generated by G contains tree t over Σ iff it holds for every node n of $T(t)$ with label l (including $\rtimes$) that n's string of $\lhd_T$-daughters is a member of $S(l)$. ⌟

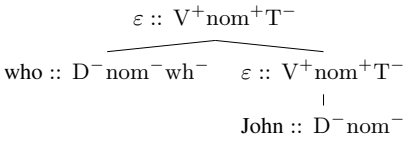


Figure 5: nom-tier of the MG dependency tree in Fig. 1; the root marker $\rtimes$ and the bottom markers $\ltimes$ are omitted.

As a concrete example, the dependency tree in Fig. 1 would yield the tier in Fig. 5 if T contains all and only those lexical items that carry an instance of nom^{-} or nom^{+}.

The dependency trees of standard MGs in SMNF are the intersection of multiple TSL tree languages (cf. Graf, 2018). One tier is needed to enforce the constraints (Final) and (Merge). This tier is trivial in the sense that it is identical to the original tree.

Definition 2 (Merge language). Let M be some MG. We define a TSL-2[TSL] grammar $G :=$

$\langle M, M, S\rangle$, where S is given by two cases. The image of $\rtimes$ under S is the set of all lexical items in M whose category feature is C^-. For each lexical item $l \in M$ such that l contains exactly n selector features $\mathrm{F}_1^+ \cdots \mathrm{F}_n^+$, $S(l)$ is the set of all strings w over M of length n such that it holds for all $1 \leq i \leq n$ that the i-th symbol in w^{-1} (the reverse of w) has category feature F_i^-. We call $L(G)$ the *Merge language of* G (denoted $Merge_G$). ⌟

In addition, each movement feature, e.g. wh or nom, also defines its own TSL tree language (assuming the MG is in SMNF).

Definition 3 (Move language). Let M be some MG in SMNF. For each $f \in Move$, we define a TSL-2[TSL-2] grammar $G_f := \langle M, T_f, S_f\rangle$. We use f^- as shorthand for any $l \in M$ that carries f^-, and $\neg\mathrm{f}^-$ for any $l \in M \cup \{\rtimes, \ltimes\}$ that does not carry f^-. Then the SL-2 function S_f is given by two cases:

$$S_f(l) := \begin{cases} (\neg\mathrm{f}^-)^*\mathrm{f}^-(\neg\mathrm{f}^-)^* & \text{if } l \text{ carries } \mathrm{f}^+, \\ (\neg\mathrm{f}^-)^* & \text{otherwise.} \end{cases}$$

We call $L(G_f)$ the *f-Move language of* G (denoted $Move_G^f$). The *Move language of* G ($Move_G$) is the intersection of all $Move_G^f$ for $f \in Move$. ⌟

Each $Move_G^f$ ensures that every lexical item carrying f^+ has exactly one daughter carrying f^-, and every lexical item carrying f^- has a mother carrying f^+. Hence $Move_G$ ensures a strict one-to-one matching between all licensor and licensee features.

Theorem 1. *For every MG G in SMNF, the intersection of $Merge_G$ and $Move_G$ contains all and only those trees that are models of all four constraints in Fig. 3.*

Due to space constraints, we do not give a formal proof of this result here. The reader is referred to Graf (2018) for the general intuition.

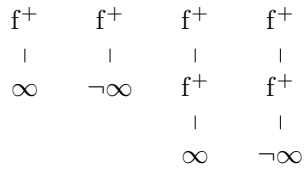
3.2 Adding persistence is easy/impossible

When we modified the first-order definition of MGs, we saw that persistent licensor features are easier to add than persistent licensee features. The former only require weakening "exactly one" to "at least one", whereas the latter involve the fairly complex notion of extended occurrences. The same split arises under the TSL view of MGs (in SMNF), and it is in fact even more pronounced. Whereas persistent licensor features are easy to add, persistent licensee features are impossible to capture with TSL-2[TSL] (or any TSL-2[X], in fact).

As before, we consider persistent licensor features first. All we have to do is to amend $S_f(n)$ in Def. 3 with a third case: if n carries persistent f^+_∞, then $S_f(n) := \big((\neg\mathrm{f}^-)^*\mathrm{f}^-(\neg\mathrm{f}^-)^*\big)^+$. This ensures that every persistent f^+ has at least one mover among its daughters on the tier, but possibly more. In other words, the persistent f^+ can check any number of f^- from distinct lexical items, but it must check at least one.

Note that $(\neg\mathrm{f}^-)^*\mathrm{f}^-(\neg\mathrm{f}^-)^*$ and $\big((\neg\mathrm{f}^-)^*\mathrm{f}^-(\neg\mathrm{f}^-)^*\big)^+$ are both TSL-2 string languages, so there is no discernible complexity difference between standard licensor features and persistent licensor features. In terms of subregular complexity, there is no reason to exclude persistent licensor features from MGs — the computational machinery that is needed for standard MGs in SMNF already provides everything that is needed to handle persistent licensor features.

Persistent licensee features, on the other hand, cannot be defined in tree TSL-2 at all. Consider the four example tiers below, where ∞ is persistent f^- and $\neg\infty$ is non-persistent f^-:



The first three tiers should be well-formed as each f^+ is an occurrence or extended occurrence — that is to say, each f^+ gets checked. The fourth tier, on the other hand, should be ill-formed because the higher f^+ cannot checked once the non-persistent f^- has been checked by the lower f^+. But there is no TSL-2 tree language that includes the first three tiers while excluding the latter. In order for the latter to be illicit, one of the following must be ruled out: f^+ as the daughter string of $\rtimes$, f^+ as the daughter string of f^+, or $\neg\infty$ as the daughter string of f^+. Ruling out one of those options would necessarily render one of the well-formed tiers illicit. This argument is a tree analogue of the fact that TSL string languages must have tier languages that are suffix substitution closed (De Santo and Graf, 2019).

In order to avoid the issue, we need a more powerful notion of TSL where the function S determines the set of licit daughter configurations for a

node n based on the whole tier context of n, rather than just the label of n. In the specific case at hand, $S(n)$ must be sensitive to the label of the tier mother of n, too: if an f^+ has an f^+ mother, its daughter string cannot contain a non-persistent f^-, while persistent f^- or no f^- at all would both be fine (the latter case allows for unbounded recursion, but as long as f^+ cannot be the tier mother of $\ltimes$ we are guaranteed to encounter a persistent f^- eventually). There are many different ways this could be formalized, yielding vastly different classes of tree languages, and we see no reason to pick a particular option at this point. The important point is that the standard notion of TSL tree languages that suffices for movement in SMNF MGs is also enough for persistent licensor features, but not for persistent licensee features.

There is one interesting caveat, though: In an MG where all licensee features are persistent, movement is again TSL-2. In this case, the only constraints on nodes carrying f^+ is that their string of daughters must not contain more than one f^- carrier and the string must not be $\ltimes$. For persistent licensor features, the first half of the constraint can also be dropped, leaving us with the simple requirement that their string of tier daughters must not be $\ltimes$. This indirectly ensures that the leafs of a movement tier must still be lexical items carrying f^-. In a certain sense, then, the problem with persistent licensee features is not so much the persistence as such, but that the availability of persistent as well as non-persistent licensee features introduces an ambiguity that cannot be resolved by considering only a node and its string of tier daughters. One can either provide a sufficiently large context for disambiguation, or avoid the ambiguity by allowing for only one type of licensee feature.

As far as we can tell, many syntactic analyses are incompatible with MGs where all licensee features are persistent, so we maintain our original claim that persistent licensee features pose a qualitatively different, more complex challenge than persistent licensor features.

4 Conclusion

We have given a rigorous first-order definition of MGs with persistent features, and based on this we have argued that not all persistent features are alike. There is a complexity difference between persistent licensor features on the one hand and persistent licensee features on the other. Persistent licensor features are unremarkable from the subregular perspective of TSL. This shows that MGs can be expanded to handle phenomena such as multiple wh-movement while preserving the core properties of their abstract tree languages. Persistent licensee features, on the other hand, introduce additional complexity, which favors the approach of Kobele (2006), where successive cyclic movement is not feature triggered and is instead a by-product of the spell-out mapping from the abstract trees to phrase structure trees.

This paper has focused on the impact of persistent features on the abstract tree language, leaving open how they affect the spell-out mapping. If two lexical items of l and l' share the same occurrence, then it is not clear how the subtrees of l and l' should be linearized with respect to each other. The feature calculus no longer provides all the necessary information. However, the MG dependency tree might. For instance, the linearization could reflect the lexicographic order of l and l' in the dependency tree, or it might be based on morphological cases (e.g. nominative preceding accusative, which precedes dative). These options are easily defined in first-order logic, which entails that they all preserve the weak generative capacity of MGs. But it is unclear which option is the empirically correct choice. Since subregular tree transductions are still severely understudied, it is unclear how these different options might change the subregular complexity of the mapping, and more work is needed in this area.

Acknowledgments

The work reported in this paper was supported by the National Science Foundation under Grant No. BCS-1845344. We thank the participants of Stony Brook's Mathematical Linguistics Reading Group, where the issue with persistent licensee features was first discussed. We also thank anonymous reviewers for their comments, which prompted several additions to this paper.

References

Rolf Backofen, James Rogers, and K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information*, 4:5–39.

Željko Bošković. 2002. On multiple wh-fronting. *Linguistic Inquiry*, 33:351–383.

Thomas Cornell and James Rogers. 1998. Model theoretic syntax. In *The Glot International State of the Article Book*, volume 1 of *Studies in Generative Grammar 48*, pages 101–125. Mouton de Gruyter.

Aniello De Santo and Thomas Graf. 2019. Structure sensitive tier projection: Applications and formal properties. In *Formal Grammar*, pages 35–50, Berlin, Heidelberg. Springer.

Hans-Martin Gärtner and Jens Michaelis. 2010. On the treatment of multiple-wh-interrogatives in Minimalist grammars. In Thomas Hanneforth and Gisbert Fanselow, editors, *Language and Logos*, pages 339–366. Akademie Verlag, Berlin.

Thomas Graf. 2012. Locality and the complexity of Minimalist derivation tree languages. In *Formal Grammar 2010/2011*, volume 7395 of *Lecture Notes in Computer Science*, pages 208–227, Heidelberg. Springer.

Thomas Graf. 2018. Why movement comes for free once you have adjunction. In *Proceedings of CLS 53*, pages 117–136.

Thomas Graf. 2020. Curbing feature coding: Strictly local feature assignment. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2020*, pages 362–371.

Thomas Graf, 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*, pages 200–215, Berlin, Heidelberg. Springer.

Thomas Graf and Aniello De Santo. 2019. Sensing tree automata as a model of syntactic dependencies. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 12–26, Toronto, Canada. Association for Computational Linguistics.

Irene Heim and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Blackwell, Oxford.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127.

Jeffrey Heinz, 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*, pages 58–64.

M. A. C. Huybregts. 1984. The weak adequacy of context-free phrase structure grammar. In Ger J. de Haan, Mieke Trommelen, and Wim Zonneveld, editors, *Van Periferie naar Kern*, pages 81–99. Foris, Dordrecht.

Adam Jardine and Kevin McMullin. 2017. Efficient learning of tier-based strictly k-local languages. In *Proceedings of Language and Automata Theory and Applications*, Lecture Notes in Computer Science, pages 64–76, Berlin. Springer.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Gregory M. Kobele. 2006. *Generating Copies: An Investigation into Structural Identity in Language and Grammar*. Ph.D. thesis, UCLA.

Gregory M. Kobele, Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to Minimalism. In *Model Theoretic Syntax at 10*, pages 71–80.

Dakotah Lambert and James Rogers. 2020. Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation and Linguistics*, volume 3.

Sabine Laszakovits. 2018. Case theory in Minimalist grammars. In *Proceedings of Formal Grammar 2018*, pages 37–61, Berlin. Springer.

Kevin McMullin, Alëna Aksënova, and Aniello De Santo. 2019. Learning phonotactic restrictions on multiple tiers. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 377–378.

Jens Michaelis and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Artifical Intelligence*, pages 329–345. Springer.

Uwe Mönnich. 2006. Grammar morphisms. Ms. University of Tübingen.

Frank Morawietz. 2003. *Two-Step Approaches to Natural Language Formalisms*. Walter de Gruyter, Berlin.

Daniel Radzinski. 1991. Chinese number names, tree adjoining languages, and mild context sensitivity. *Computational Linguistics*, 17:277–300.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–345.

Edward P. Stabler. 1997. Derivational Minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer, Berlin.

Edward P. Stabler. 2011. Computational perspectives on Minimalism. In Cedric Boeckx, editor, *Oxford Handbook of Linguistic Minimalism*, pages 617–643. Oxford University Press, Oxford.

Edward P. Stabler and Edward Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science*, 293:345–363.

Mai Ha Vu. 2018. Towards a formal description of NPI-licensing patterns. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, volume 1, pages 154–163. Article 17.

Mai Ha Vu, Nazila Shafiei, and Thomas Graf. 2019. Case assignment in TSL syntax: A case study. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 267–276.