Locality and the Complexity of Minimalist Derivation Tree Languages

Thomas Graf

Department of Linguistics
University of California, Los Angeles
tgraf@ucla.edu
http://tgraf.bol.ucla.edu

Abstract. Minimalist grammars provide a formalization of Minimalist syntax which allows us to study how the components of said theory affect its expressivity. A central concern of Minimalist syntax is the locality of the displacement operation Move. In Minimalist grammars, however, Move is unbounded. This paper is a study of the repercussions of limiting movement with respect to the number of *slices* a moved constituent is allowed to cross, where a slice is the derivation tree equivalent of the phrase projected by a lexical item in the derived tree. I show that this locality condition 1) has no effect on weak generative capacity 2) has no effect on a Minimalist derivation tree language's recognizability by top-down automata 3) renders Minimalist derivation tree languages strictly locally testable, whereas their unrestricted counterparts aren't even locally threshold testable.

Key words: Minimalist grammars, locality, subregular tree languages, first-order logic, top-down tree automata

Introduction

Even though Minimalist grammars (MGs) weren't introduced by Stabler [16] with the sole intent of scrutinizing the merits of ideas put forward by syntacticians in the wake of Chomsky's Minimalist Program [2], a lot of work on MGs certainly focuses on this aspect [cf. 17]. Recently, considerable attention has also been directed towards the role played by derivation trees in MGs [4, 7, 8]. It is now known that every MG's derivation tree language is regular and "almost" closed under intersection with regular tree languages (some refinement of category labels is usually required), but it is still an open question which class of tree languages approximates them reasonably well. This paper combines both research strands by taking the linguistically motivated restriction to local movement as its vantage point for an examination of the structural complexity of Minimalist derivation tree languages (MDTLs). The main result is that while bounding the distance of movement leaves weak generative capacity unaffected, the complexity of MDTLs is lowered to a degree where they become strictly locally testable. Since MGs are fully characterized by their MDTLs, lowering the

upper bound from regular to strictly locally testable may prove useful for various practical applications operating directly on the derivation trees, in particular parsing [9, 18, 21].

The paper is laid out as follows: After some general technical remarks, Sec. 2 introduces MGs, their derivation trees, and the important concept of slices. Building on these notions, I define movement-free and k-local MGs, and I prove that every MG can be converted into a k-local one. The complexity of these derivation trees is then studied with respect to several subregular languages classes in Sec. 3. I first show that MDTLs can be recognized by l-r-deterministic top-down tree automata, but not by sensing tree automata, which entails non-recognizability by deterministic top-down tree automata. Furthermore, every k-local MG G has a strictly locally κ-testable derivation tree language, with the value of κ depending on several parameters of G. This result is followed by a demonstration that unrestricted MDTLs are not locally threshold testable. However, they are definable in first-order logic with predicates for left child, right child, proper dominance, and equivalence (Sec. 4).

1 Preliminaries and Notation

As usual, $\mathbb{N}$ denotes the set of non-negative integers. A *tree domain* is a finite subset D of $\mathbb{N}^*$ such that, for $w \in \mathbb{N}^*$ and $j \in \mathbb{N}$, $wj \in D$ implies both $w \in D$ and $wi \in D$ for all $i < j$. Every $n \in D$ is called a *node*. Given nodes $m, n \in D$, m *immediately dominates* n iff $n = mi$, $i \in \mathbb{N}$. In this case we also say m is the mother of n, or conversely, n is a daughter of m. The transitive closure of the immediate dominance relation is called *(proper) dominance*. A node that does not dominate any other nodes is a *leaf*, and the unique node that isn't dominated by any nodes is called the *root*.

Now let Σ be a *ranked* alphabet, i.e. every $\sigma \in \Sigma$ has a unique non-negative *rank* (*arity*); $\Sigma^{(n)}$ is the set of all n-ary symbols in Σ. A Σ-tree is a pair $T := \langle D, \ell \rangle$, where D is a tree domain and $\ell : D \to \Sigma$ is a function assigning each node n a *label* drawn from Σ such that $\ell(n) \in \Sigma^{(d)}$ iff n has d daughters. Usually the alphabet will not be indicated in writing when it is irrelevant or can be inferred from the context. Sometimes trees will be given in functional notation such that $f(t_1, \ldots, t_n)$ is the tree whose root node is labeled f and immediately dominates trees $t_1, \ldots, t_n$. I denote by T_Σ the set of all trees such that for $n \geq 0$, $f(t_1, \ldots, t_n)$ is in T_Σ iff $f \in \Sigma^{(n)}$ and $t_i \in T_\Sigma$, $1 \leq i \leq n$. A *tree language* is some subset of T_Σ.

A *context* C is a $\Sigma \cup \{\Box\}$-tree, where $\Box$ is a new symbol that appears on exactly one leaf of C, designating it as the *port* of C. A context C with $\Box$ occurring in the configuration $c := \sigma(t_1, \ldots, \Box, \ldots, t_n)$, $\sigma \in \Sigma^{(n)}$ and each t_i a Σ-tree, can be composed with context C' (written $C \cdot C'$) by replacing c with $\sigma(t_1, \ldots, C', \ldots, t_n)$. This extends naturally to all cases where $C = \Box$ and C' is a tree rather than a context. Given a Σ-tree $t := \langle D, \ell \rangle$ and some node u of t, $t|_u := \langle D|_u, \ell \rangle$ denotes the subtree rooted by u in t, such that $D|_u := \{n \in D \mid u = n \text{ or } u \text{ dominates } n\}$ and dominance and the labeling function are

preserved. For any tree t with nodes m and n of t such that either $m = n$ or m dominates n, $C_t[m, n)$ is the context obtained from $t|_m$ by replacing $t|_n$ by a port. If s and t are trees, r the root of s and u some node of s, then $s[u \leftarrow t] := C_s[r, u) \cdot t$.

Let m and n be nodes of some tree t. A *path* from m to n is a sequence of node $\langle i_0, \ldots, i_k \rangle$ such that $i_0 = m$, $i_k = n$, and for all $j < k$, i_j is the mother or the daughter of i_{j+1}. A path containing k nodes is of *length* $k - 1$. The *distance* between nodes m and n is the length of the shortest path from m to n. The *depth* of a tree t is identical to the greatest distance between the root of t and one of its leafs.

We now move on to defining the strictly locally testable languages, following the exposition in [21]. For each Σ-tree and choice of $k \geq 1$, we define its *k-factors*, or more precisely, its *k-prefixes*, *k-forks* and *k-suffixes* as follows:

$$p_k(\sigma(t_1, \ldots, t_n)) := \begin{cases} \sigma & \text{if } k = 1 \text{ or } \sigma \text{ has no children} \\ \sigma(p_{k-1}(t_1), \ldots, p_{k-1}(t_n)) & \text{otherwise} \end{cases}$$

$$f_k(\sigma(t_1, \ldots, t_n)) := \begin{cases} \emptyset & \text{if } \sigma(t_1, \ldots, t_n) \text{ is of depth } d < k - 1 \\ \{p_k(\sigma(t_1, \ldots, t_n))\} \cup \bigcup_{i=1}^{n} f_k(t_i) & \text{otherwise} \end{cases}$$

$$s_k(\sigma(t_1, \ldots, t_n)) := \begin{cases} \{\sigma(t_1, \ldots, t_n)\} \cup \bigcup_{i=1}^{n} s_k(t_i) & \text{if } \sigma(t_1, \ldots, t_n) \text{ is of depth } d < k - 1 \\ \bigcup_{i=1}^{m} s_k(t_i) & \text{otherwise} \end{cases}$$

A tree language $L \subseteq T_\Sigma$ is *strictly locally k-testable* (in SL_k) iff there exist three finite subsets R, F, and S, such that $t \in L$ iff $p_{k-1}(t) \in R$, $f_k(t) \subseteq F$, and $s_{k-1} \subseteq S$. A language is *local* (in LOC) iff it is in SL_2. It is locally k-threshold testable (in LTT_k) iff furthermore each k-factor must appear a specific number of times, counting up to some fixed threshold. When the threshold is set to 1, L is locally k-testable (in LT_k). We say that L belongs to one of these classes iff there is some k such that L is k-testable in the intended sense. Finally, L is *regular* iff it is the range of a transduction computed by some linear tree transducer with domain T_Σ (the reader is referred to [5] for further details).

Definition 1. *A* linear tree transducer *is a 5-tuple $A := \langle \Sigma, \Omega, Q, Q', \Delta \rangle$, where Σ and Ω are finite ranked alphabets, Q is a finite set of states, $Q' \subseteq Q$ the set of final states, and Δ is a set of productions of the form $f(q_1(x_1), \ldots, q_n(x_n)) \to q(t)$, where $f \in \Sigma^{(n)}$, $q_1, \ldots, q_n, q \in Q$, $t \in T_{\Omega \cup \{x_1, \ldots, x_n\}}$, and each x_i occurs at most once in t.*

It is a well-known fact that $\mathrm{LOC} \subset \mathrm{SL} \subset \mathrm{LT} \subset \mathrm{LTT} \subset \mathrm{REG}$.

2 Minimalist Grammars

2.1 Introduction and Examples

MGs are a highly lexicalized formalism. Every lexical item (LI) comes equipped with a linear sequence of features that have to be "checked", or equivalently,

"erased" in the right order. Features come in two varieties that can only be checked by the operations Merge and Move, respectively. Merge conjoins trees, while Move displaces subtrees. A very simple MG, for example, is instantiated by the following lexicon.

man :: n	the :: =n d	ε :: =v +nom t
John :: d	the :: =n d −nom	ε :: =t c
John :: d −nom	the :: =n d −top	ε :: =t +top c
John :: d −top	killed :: =d =d v	

The first component of an LI denotes its phonetic exponent, the second one its feature string. Features without a prefix represent categories (n for noun, d for determiner, and so on). A category feature, say n, of LI l is checked whenever Merge combines l with another LI l' such that the respective first unchecked features of l and l' are n and the matching selector feature =n. This is also the only feature configuration in which Merge may apply. Hence Merge could combine *the* and *man*, yielding *the man*, which in turn can be merged with *killed*, but not *the* and *John* (no compatible features at all), or *killed* and *the* (the first unchecked feature of *the* is =n, which is incompatible with the =d on *killed*). The feature combinatorics of the Move operation are essentially the same, with the only difference being that Move applies to features prefixed with + and − (licensor and licensee, respectively). Note that Merge introduces new material into the derivation, whereas Move merely displaces old material — intuitively, the subtree headed by an LI l with some licensee feature $-f$ as its first unchecked feature is moved into the specifier of the closest LI l' that c-commands l and has $+f$ as its first unchecked feature.

An utterance is well-formed if it can be assigned a derivation in which all features were checked except for the category feature of the last LI to be merged, which must be a so-called final category (usually c). The MG above generates the following eight sentences, and only those (assuming that c is the only final category):

(1) a. John/The man killed John/the man.
b. John/The man, John/the man killed.

A derivation tree for one of the sentences with topicalization is given in Fig. 1.

Despite its simplicity, the feature calculus controlling Move allows for a dazzling array of movement configurations, in particular *remnant movement*, in which some XP is extracted from some YP via Move before YP itself is moved to a higher position. Remnant movement allows for an elegant reanalysis of cases where apparently non-phrasal constituents end up in positions reserved for phrases, such as in the German example below.

(2) [$_{\text{CP}}$ Geküsst$_i$ hat$_j$ [$_{\text{TP}}$ der Hans die Maria t_j t_i.]]
kissed has the John the Mary.
'John kissed Mary.'

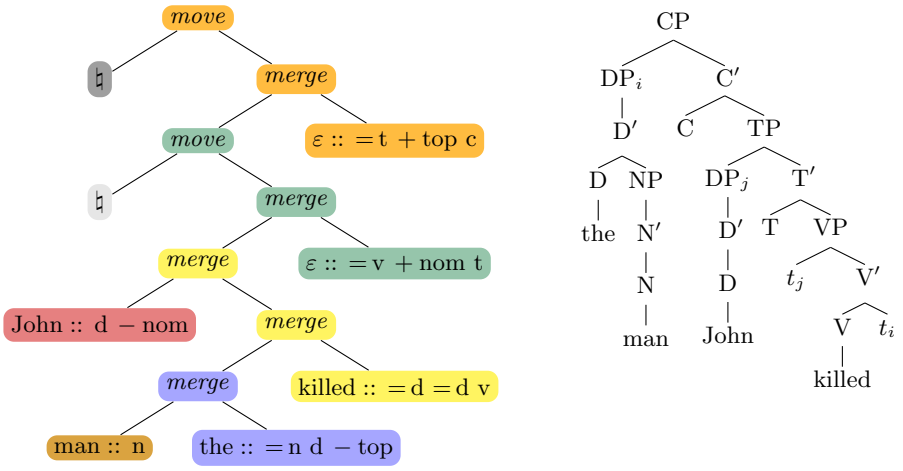

Fig. 1. Left: derivation tree of *The man, John killed*, depicted in the final format adopted in this paper and with slices indicated by color; Right: Corresponding X′-tree

Instead of having just the V-head *geküsst* move into SpecCP (which is at odds with standard assumptions about phrase structure), one can fall back to a remnant movement analysis: the object moves out of the VP, followed by movement of the remaining VP into SpecCP. At this point the VP's only phonetic exponent is its V-head, so that the end result is indistinguishable from the scenario where only the V-head had undergone movement. For our purposes, remnant movement is of interest because of the crucial role it plays in the proof that every instance of Move that spans arbitrary distances can be decomposed into a sequence of local Move steps (Thm. 1).

2.2 Minimalist Grammars, Derivation Trees, and Slices

As the focus of this paper is on the derivation trees of MGs rather than the phrase structure trees derived via Merge and Move, the details of both operations are of interest to us only in so far as they have ramifications for the shape of derivation trees or the string yield (which will be important for Thm 1). Nonetheless I give a full definition of the formalism here, staying close to the chain-based exposition of [19]. After that I formally define MDTLs and introduce the notion of slices.

Definition 2. *A* Minimalist grammar *is a 6-tuple* $G := \langle \Sigma, Feat, F, Types, Lex, Op \rangle$*, where*

- $\Sigma \neq \emptyset$ *is the alphabet,*
- $Feat$ *is the union of a non-empty set* Base *of* basic features *(also called* category features*) and its prefixed variants* $\{=f \mid f \in \text{Base}\}$, $\{+f \mid f \in \text{Base}\}$, $\{-f \mid f \in \text{Base}\}$ *of* selector, licensor, *and* licensee *features, respectively,*
- $F \subseteq$ Base *is a set of final categories,*

- *Types* $:= \{::, :\}$ *distinguishes* lexical *from* derived *expressions,*
- *the lexicon Lex is a finite subset of* $\Sigma^* \times \{::\} \times Feat^*$*,*
- *and Op is the set of generating functions to be defined below.*

A chain *is a triple in* $\Sigma^* \times Types \times Feat^*$*, and C denotes the set of all chains (whence* $Lex \subset C$*). Non-empty sequences of chains will be referred to as* expressions*, the set of which is called E.*

The set Op of generating functions consists of the operations merge and move, which are the respective unions of the following functions, with $s, t \in \Sigma^*$*,* $\cdot \in Types$*,* $f \in$ BASE*,* $\gamma \in Feat^*$*,* $\delta \in Feat^+$*, and chains* $\alpha_1, \ldots, \alpha_k$*,* $\iota_1, \ldots, \iota_k$*,* $0 \leq k, l$*:*

$$\frac{s :: = f\gamma \qquad t \cdot f, \iota_1, \ldots, \iota_k}{st : \gamma, \iota_1, \ldots, \iota_k} \; merge1$$

$$\frac{s : = f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f, \iota_1, \ldots, \iota_l}{ts : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l} \; merge2$$

$$\frac{s \cdot = f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f\delta, \iota_1, \ldots, \iota_l}{s : \gamma, \alpha_1, \ldots, \alpha_k, t : \delta, \iota_1, \ldots, \iota_l} \; merge3$$

$$\frac{s : +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \ldots, \alpha_k}{ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \alpha_k} \; move1$$

$$\frac{s : +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f\delta, \alpha_{i+1}, \ldots, \alpha_k}{s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k} \; move2$$

Furthermore, all chains must satisfy the Shortest Move Constraint (SMC), according to which no two chains in the domain of move display the same licensee feature $-f$ *as their first feature. The string language generated by G is* $L(G) := \{\sigma \mid \langle \sigma \cdot c \rangle \in closure(Lex, Op), \cdot \in Types, c \in F\}$*.*

MGs and MCFGs [15] have the same weak generative capacity [6, 12]. In fact, for every MG there exists a strongly equivalent MCFG, but not the other way round. In a certain sense, then, one may view MGs as a narrowly restricted way of specifying MCFGs. A more peculiar fact about MGs is that in order for $l \in Lex$ to occur in a well-formed derivation, its feature string must be in $\{+f, =f \mid f \in \text{BASE}\}^* \times \text{BASE} \times \{-f \mid f \in \text{BASE}\}^*$. I will implicitly invoke this fact several times throughout the paper.

I now turn to the derivation trees of an MG, defining them in two steps.

Definition 3. *Given an MG* $G := \langle \Sigma, Feat, F, Types, Lex, Op \rangle$*, the largest subset of* T_E *satisfying the following conditions is called the* string-annotated derivation tree language $\text{sder}(G)$ *of G:*

- *For every leaf node n,* $\ell(n) = \langle l \rangle$*,* $l \in Lex$*.*
- *For every subtree* $m(d_1, \ldots, d_n)$*,* $n \geq 1$*,* $op(d_1, \ldots, d_n)$ *is defined for exactly one* $op \in \{merge, move\}$ *and* $\ell(n) = op(d_1, \ldots, d_n)$*.*
- *For n the root node,* $\ell(n) = \langle \sigma : c \rangle$*, where* $\sigma \in \Sigma^*$ *and* $c \in F$*.*

Definition 4. *Given an MG G, its* Minimalist derivation tree language mder(G) *is the set of trees obtained from* sder(G) *by the map* μ*:*

- $\mu(\langle l \rangle) = l$*, where* $l \in Lex$
- $\mu(e(e_1, \ldots, e_n)) = op(\mu(e_1), \ldots, \mu(e_n))$*, where* $e, e_1, \ldots, e_n \in E$*,* $n \geq 1$*, and op is the unique operation in Op such that* $op(e_1, \ldots, e_n) = e$

As was first observed in [8], every MDTL is regular. The basic idea is to equip a bottom-up tree automaton with states corresponding to the feature components of the string-annotated derivation trees (there are only finitely many thanks to the SMC) and have its transition rules recast the conditions imposed on Merge and Move by the feature calculus. Interestingly, an MG's set of derived trees — which is not regular — can be obtained from its MDTL in an efficient way using a multi bottom-up tree transducer. This in turn means that MDTLs are the key to capturing MGs by finite-state means.

In [4], the concept of slices is introduced. Intuitively, slice(l) is the derivation tree equivalent of the phrase projected by l in the derived tree (using the standard linguistic notion of projection). That is to say, slices mark the subpart of the derivation that l has control over by virtue of its selector and licensor features (cf. Fig. 1 on page 5). From this perspective a derivation tree language is simply the result of combining slices in all possible ways such that none of conditions imposed on Merge and Move by the feature calculus are violated.

Definition 5. *Given a Minimalist derivation tree* $T := \langle D, \ell \rangle$ *and LI* l *occurring in T, the* slice of l *is the pair* slice(l) $:= \langle S, \ell \rangle$*, where* $S \subseteq D$*,* $l \in S$*, and if node* $n \in D$ *immediately dominates a node* $s \in S$*, then* $n \in S$ *iff the operation denoted by* $\ell(n)$ *erased a selector or licensor feature on* l*. The unique* $n \in S$ *that is not dominated by any* $n' \in S$ *is called the* slice root *of* l*.*

The following properties hold of slices for every MG G [cf. 4]:

- Let $|\gamma|$ denote the length of the longest γ such that there is some $l \in Lex_G$ with feature string $\gamma c \delta$, $\gamma, \delta \in Feat^*$, $c \in$ Base. Then for every $t \in$ mder(G) and slice $s := \langle S, \ell \rangle$ of t, $1 \leq |S| \leq |\gamma| + 1$.
- Every tree $t \in$ mder(G) is partitioned into slices.
- All slices are continuous (i.e. if node y is the child of x and the mother of z such that x and z belong to the same slice s, then y also belongs to s).

As the order of siblings is irrelevant in derivation trees, I stipulate for the sake of simplicity that all slices are strictly right-branching. Moreover, I assume that derivation trees are strictly binary branching, as this simplifies the math at various points in this paper — a small change which is easily accommodated by mapping $move(t)$ to $move(\natural, t)$, where $\natural$ is a new symbol not in $Lex \cup Op$.

Now given an LI l, node n is the k^{th} node of slice(l) iff the shortest path from n to l is the sequence $\langle n_1, \ldots, n_k, l \rangle$ such that $n = n_1$ and no n_i is a left daughter, $1 \leq i \leq k$. Furthermore, n is associated to feature f iff n is the i^{th} node of slice(l) and f is the i^{th} feature of l. Two features g and h are said to *match* iff there is a feature $f \in$ Base such that either $g = +f$ and $h = -f$ or $g = = f$ and $h = f$. Finally, a node n matches an LI l iff for some feature g of l, n is associated to a feature matching g.

2.3 Locality and Subclasses of Derivation Tree Languages

If one builds on the notion of slices, a natural way of imposing locality conditions on movement suggests itself. Take any MG G, $t \in \text{mder}(G)$, and suppose that l is an LI occurring in t with licensee features $-f_1, \ldots, -f_n$. Let $move[l] := \langle move_1, \ldots, move_n \rangle$ be the sequence of Move nodes in t such that the operation denoted by $move_i$ checked feature $-f_i$ on l, $1 \leq i \leq n$. An MG G is *k-local* or of *locality rank k*, $k \geq 1$, iff it holds for every $t \in \text{mder}(G)$ and LI l occurring in t with $move[l] := \langle move_1, \ldots, move_n \rangle$ that no more than $k-1$ slices intervene between m_i and m_{i+1} in $\langle m_0, m_1, \ldots, m_n \rangle := \langle l \rangle \cdot move[l]$, $0 \leq i < n$. Equivalently, the shortest path from $move_i$ to $move_{i+1}$ may contain at most k left branches. If G is k-local, we also say that movement in G is k-local or k-bounded. An MG G is *movement-free* iff no tree in $\text{mder}(G)$ contains a node labeled *move*. It is *unrestricted* iff it is neither movement-free nor k-local for any $k \geq 1$. This terminology extends to MDTLs in the natural way. By $\text{MDTL}[merge]$ and $\text{MDTL}[merge, move^{(k)}]$ I denote, respectively, the class of all movement-free MDTLs and the class of all k-local MDTLs. The class of all MDTLs is simply denoted by $\text{MDTL}[merge, move]$.

The notion of k-boundedness is motivated by the linguistic assumption that movement is successive-cyclic. In (3), for instance, the wh-phrase may not move from its base position to the beginning of the utterance in one fell swoop but rather has to land in every CP.

(3) Which professor$_i$ did John say [$_{\text{CP}}$ t_i that Bill told him [$_{\text{CP}}$ t_i that Mary has a crush on t_i]]

This assumption can be used to explain various syntactic, semantic and even morphological phenomena, for instance embedded inversion, stranding of quantifiers, binding ambiguities and complementizer agreement [cf. 3]. The following contrast provides a very simple example.

(4) a. [$_{\text{CP}_2}$ When$_i$ did John t_i tell you [$_{\text{CP}_1}$ who$_j$ Mary will meet t_j?]]
b. * [$_{\text{CP}_2}$ When$_i$ did John tell you [$_{\text{CP}_1}$ who$_j$ Mary will t_i meet t_j?]]

In (4a), *when* originates in the matrix clause and moves directly to the corresponding SpecCP position, while *who* does the same in the embedded clause. In (4b), *when* is supposed to originate in the embedded clause together with *who*, but now they cannot both move into CP-specifiers. In one case, *who* moves first, so that it fills SpecCP_1. As a consequence, *when* cannot move to SpecCP_2 due to successive cyclicity requiring it to go through the SpecCP_1 first, which is already filled by *who*. In the other case, *where* moves first but on its way to SpecCP_2 it leaves a trace in SpecCP_1 so that the latter is no longer available as a landing site to *who*. The details of the analysis have changed significantly over the years, and recently it has even been argued that CPs do not matter for cyclicity [3], but the basic idea that long-distance movement is in fact a sequence of local movement steps has remained unaltered.

If one ignores adjuncts, the kind of cyclicity involved in the examples above can be reinterpreted as wh-movement being 3-local: every clause consists of the

slices CP, TP, VP, so movement from one CP to the next creates paths containing 3 left branches. The analogy between successive-cyclic movement and the restriction to k-locality is far from perfect, of course. Among other things, successive-cyclic movement is relativized to specific positions, whereas k-locality only cares about distance. But if the value of k is carefully chosen and combined with certain regular constraints on derivation trees [4, 7] that prevent movement from skipping, say, CP slices, a reasonably close approximation can be achieved.

Surprisingly, every MG can be translated into a weakly equivalent 1-local one. The idea is that instead of having subtree s move directly to its target position t, s can hitch a ride by being selected by another LI l. As long as the string component of l is empty, this will have no effect on the string yield. This reduces unbounded movement to k-local movement, which in turn can easily be reduced to 1-local movement.

Theorem 1. *For every unrestricted MG G, there is an MG G' of locality rank 1 that defines the same string language.*

Proof. I sketch two linear (bottom-up) transducers τ and τ'. The former does most of the work by translating every $t \in \text{mder}(G)$ into its corresponding $\max(E)$-local t', where $\max(E)$ is the maximum length of expressions generated by G (i.e. the maximum number of chains per expression). The latter then rewrites t' as the 1-local t''. Since MDTLs are regular, and the image of a regular language under a linear transduction is itself regular, the output of $\tau \cdot \tau'$ is, too. This set is then intersected with $\text{mder}(G'')$, where $Lex_{G''} := \Omega_{\tau'} \setminus Op_{G''}$ and $F_{G''} := F_G$. The result of this intersection can then be automatically converted into a new MG, our G' [4, 7].

The idea underlying τ is that unbounded movement can be localized by creating intermediary landing sites which have no effect on the string language. The main task of the transducer is to insert those intermediate landing sites and transfer (a subset of) the features of each moving element to its landing site. By definition there are at most $n = \max(E) - 1$ moving elements at any given point in the derivation. Each state of τ consists of 1) n components q_i that memorize the feature string of the moving items and which of them have already been reinstantiated, 2) the expression the current node evaluates too (modulo the string component), 3) a boolean flag b that indicates whether some LI had the new licensee feature $-s_0$ added to its feature string.

Suppose we are at a node in the derivation tree t that belongs to the slice of LI $l := \sigma :: \gamma c \delta$ and that there are $0 \leq m \leq n$ moving LIs $l_i := \sigma :: \gamma_i c_i \delta_i$, where $1 \leq i \leq m$ and $c, c_i \in \textsc{Base}$ and $\sigma, \sigma_i \in \Sigma$ and δ_i is of the form $-f_{i_1}, \ldots, -f_{i_k}$, $k \geq 1$. For each i and $1 \leq u \leq v \leq k$, we define new LIs $l_i^c[u, v] := \varepsilon :: =c + f_{i_u}\ c - s_i - f_{i_u}\ \delta_i[u{+}1, v]$, where $-s_i$ is a new licensee feature and $\delta_i[u+1, v] := -f_{i_{u+1}}, \ldots, -f_{i_v}$. Given a feature string $\phi := f_1, \ldots, f_k$, we furthermore let $q(j, \phi) := f_1, \ldots, f_j \bullet f_{j+1} \ldots f_k$, $0 \leq j \leq k$.

The transducer τ now has to perform the following steps: First, if l is not itself among the moving elements, τ non-deterministically replaces it by $\hat{l} := \varepsilon :: \gamma c - s_0$ and switches the boolean flag b in its state from 0 to 1. If either

$m = 0$ and $b = 1$ or $m \geq 1$ and $b = 0$, τ aborts at the slice root of $\hat{l}/l$. Second, τ replaces each l_i carrying more than one licensee feature by $\hat{l}_i := \sigma_i :: \gamma c_i - f_{i_1}$ and stores $q(0, \delta_i[1,k])$ in q_i. Third, when τ reaches the slice root of $l/\hat{l}$, it inserts $C(l^c_m[u_m, v_m]) \cdot \ldots \cdot C(l^c_1[u_1, v_1])$, where $C(l^c_i[u_i, v_i]) := \mathit{move}(\natural, \mathit{merge}(\Box, l^c_i[u_i, v_i]))$, and u_i must be such that $q(u_i - 1, \delta_i[1,k])$ is the string stored in q_i. The value of v_i is chosen non-deterministically — in order for τ not to abort it must hold that every f_{i_j} can get checked later on without further intermediate movement sites for all $j \leq v_i$ but not for $j = v_i + 1$ (this is easily verified, as τ keeps track of licensor features in the second component of its states). With the insertion of $C(l^c_i[u_i, v_i])$, q_i is updated to $q(v_i, \delta_i[1,k])$. So far then, τ has introduced new slices $\mathrm{slice}(l^c_i[u_i, v_i])$ that function as the respective landing sites for each l_i, or rather, their impoverished counterpart $\hat{l}_i$. The fourth step requires τ to insert the context $C(s^c[m,b])$ above $C(l_m[u_m, v_m])$, where $s^c[j,b] := \varepsilon :: {=}c \ + s_{1-b} \ldots + s_j \ c$ for $1 \leq j \leq n$, and $C(s^c[j,b]) := \underbrace{\mathit{move}(\natural, \Box) \cdot}_{j+b \text{ times}} \mathit{merge}(\Box, s^c[j,b])$.

This enforces remnant movement of $l/\hat{l}$ and all $\hat{l}_i$, allowing each $\mathrm{slice}(l^c_i[k])$ to move freely later on without carrying along any other parts of the derived tree (which would induce a change in the string yield). The procedure as outlined above is iterated (with the value of c varying with l) until no more features need to be checked off. Since there are at most $n = \max(E) - 1$ moving elements in G, no LI l_i (including $l/\hat{l}$) has to cross more than n slices in order to check its $-f_{i_1}$ feature against $l^c_i[u, v]$ or its $-s_i$ feature against $s^c[j,b]$. Thus every instance of *move* is $\max(E)$-bounded.

All instances of $(k+1)$-bounded movement, $1 \leq k \leq \max(E)$, can be made 1-local as follows. Assume that slices $\mathrm{slice}(l_1), \ldots \mathrm{slice}(l_k)$ intervene between $l := \sigma :: \gamma c \delta$ and its next occurrence. Then τ' has to prefix δ with k new movement licensee features $-l_1 \ldots - l_k$, and for each $l^c_i[u, v]$ $(1 \leq i \leq k)$ add $+l_i$ to the end of its feature string and $\mathit{move}(\natural, \Box)$ above its slice root. If several LIs move through a slice, the number of licensor features and Move nodes has to be adapted accordingly. Crucially, both the number of moving elements and the distance between LIs and individual occurrences is finitely bounded, so this strategy can easily be carried out by a non-deterministic linear transducer. □

3 (Un)Definability in Some Subregular Language Classes

3.1 Deterministic Top-Down Automata

Since MDTLs are regular, they can be recognized by non-deterministic top-down tree automata [cf. 5]. As we will see now, top-down non-determinism can be dispensed with only if it is compensated for by unbounded look-ahead. I consider two common variants of the standard deterministic top-down tree automaton (DTDA), both of which are more powerful than DTDAs but do not recognize all regular languages. One is the *sensing tree automaton* (STA) [11], which may also take the labels of a node's children into account in order to decide which

states should be assigned to them, while the other is the l-r-deterministic DTDA (lrDTDA) [13], which allows for a limited kind of non-determinism. The classes of languages recognized by STAs and lrDTDAs are incomparable, but can easily be characterized in descriptive terms. For this reason, I focus on the languages themselves rather than the automata, and no further technical details of the latter will be discussed here (the interested reader is referred to [11] and references therein).

Definition 6. *Given a node v of some Σ-tree t,* $\mathrm{lsib}^t(u)$ *is the string consisting of the label of u's left sister (if it exists) followed by the label of u, and* $\mathrm{rsib}^t(u)$ *is the string consisting of the label of u and the label of its right sister (if it exists). Let $u_1, \ldots, u_n$ be the shortest path of nodes extending from the root to v such that u_1 is the root and $u_n = v$. Let $\blacksquare$ and $\clubsuit$ be two new symbols not in Σ. By* $\mathrm{spine}^t(v)$ *we denote the string recursively defined by*

$$\begin{aligned} \mathrm{spine}^t(u_1) &= \mathrm{lsib}^t(u_1)\blacksquare\mathrm{rsib}^t(u_1) \\ \mathrm{spine}^t(u_1, \ldots, u_n) &= \mathrm{spine}^t(u_1, \ldots, u_{n-1}) \clubsuit \mathrm{lsib}^t(u_n) \blacksquare \mathrm{rsib}^t(u_n) \end{aligned}$$

A regular tree language L is spine-closed *iff it holds for all trees $s, t \in L$ and nodes u and v belonging to s and t, respectively, that* $\mathrm{spine}^s(u) = \mathrm{spine}^t(v)$ *implies $s[u \leftarrow t|_v] \in L$.*

Definition 7. *A regular tree language L is* homogeneous *iff it holds that if $t[u \leftarrow a(t_1, t_2)] \in L$, $t[u \leftarrow a(s_1, t_2)] \in L$ and $t[u \leftarrow a(t_1, s_2)] \in L$, then also $t[u \leftarrow a(s_1, s_2)] \in L$.*

Proposition 1. *A regular tree language L is recognizable by*

- *an STA iff L is spine-closed [10].*
- *an lrDTDA iff L is homogeneous [13].*

Thanks to these characterizations, results for MDTLs are easily obtained.

Theorem 2. $\textsc{mdtl}[merge]$ *and the class of tree languages recognized by STAs are incomparable.*

Proof. Let grammar G be defined by the following LIs (with names in square brackets for reference) and $F_G := \{a, b\}$:

$$\begin{array}{lll} [a_0]\ \mathrm{a} :: \mathrm{a} & [a_1]\ \mathrm{a} :: =\mathrm{a}\ \mathrm{a} & [a_2]\ \mathrm{a} :: =\mathrm{a}\ =\mathrm{a}\ \mathrm{a} \\ [b_0]\ \mathrm{b} :: \mathrm{b} & [b_1]\ \mathrm{b} :: =\mathrm{b}\ \mathrm{b} & [b_2]\ \mathrm{b} :: =\mathrm{b}\ =\mathrm{b}\ \mathrm{b} \end{array}$$

Consider the derivation tree $t_a := merge_1(merge_2(a_0, a_1), merge_3(a_0, a_2))$ and its counterpart $t_b := merge_1(merge_2(b_0, b_1), merge_3(b_0, b_2))$ — the indices are for the reader's convenience. Even though $\mathrm{spine}^{t_a}(merge_2) = \mathrm{spine}^{t_b}(merge_2)$, it holds that $merge_1(merge_2(b_0, b_1), merge_3(a_0, a_2)) \notin \mathrm{mder}(G)$, so $\mathrm{mder}(G)$ is not spine-closed. □

Theorem 3. *Every $L \in \textsc{mdtl}[merge, move]$ is recognized by some lrDTDA.*

Proof. I show that every MDTL is homogeneous. For $a = move$, closure is trivially satisfied. So let $a = merge$. Merge depends only on the distribution of category and selector features, and there is no way to distribute these over t_1, t_2, s_1 and s_2 such that the fourth tree would be an illicit instance of Merge: In order for Merge to be licensed, one of t_1 or t_2 must have some category feature c as its first unchecked feature, and the other one the matching selector feature $=$c. Assume w.l.o.g. that t_2 carries the selector feature $=$c. Then s_1 must also have feature c, and s_2 feature $=$c. We also know that s_1 and t_1 on the one hand and s_2 and t_2 on the other agree on all features following these selector/licensor features, since the derivations differ only w.r.t. the subtree rooted by a. It follows that Merger of s_1 and s_2 is licit, so the required closure property obtains. □

Martens et al. [11] point out a peculiar property of languages recognized by lrTDAs but not by STAs: in order to determine which states should be assigned to the children of the root, one has to look arbitrarily deep into at least one of the subtrees dominated by the root. This is indeed typical of unrestricted MDTLs, where movement features at the very bottom of a derivation introduce dependencies that — given the impoverished nature of the interior node labels — cannot be predicted deterministically in a top-down fashion without unbounded look-ahead. The class of lrTDAs overshoots the mark, though, as it fails to draw a distinction even between MDTL[*merge*, *move*] and MDTL[*merge*].

3.2 Strictly Local and Locally Threshold Testable Languages

Let us now traverse the subregular hierarchy from the bottom instead, starting with LOC and subsequently moving on to SL and LTT. As lrTDAs before, local sets lack the granularity to distinguish any of the subclasses of MDTLs. But where the lrTDAs universally succeeded, local sets universally fail.

Theorem 4. MDTL[*merge*] *and the class of local sets are incomparable.*

Proof. Consider any movement-free MDTL L with a derivation containing a subtree of the form $t := merge(l, merge)$. As we require slices to be right-branching, l contains no selector or licensor features. Furthermore, the finiteness of the lexicon establishes an upper bound $|\gamma|+1$ on the size of slices. However, LOC $=$ SL$_2$, so if $L \in$ LOC, t could be composed with itself arbitrarily often, yielding slices of unbounded size. □

The shortcomings of LOC can be circumvented, though, by extending the size of the locality domain, i.e. by moving to SL$_k$ for some sufficiently large $k > 2$. Let $|\delta|$ be the maximum number of licensee features that may occur on a single LI, analogously to $|\gamma|$. Given a k-local MG, set $\kappa := (|\gamma| + 1) * (|\delta| * k + 1) + 1$.

Theorem 5. *Every* $L \in$ MDTL[*merge*, $move^{(k)}$] *is strictly* κ*-local.*

Before we may proceed to the actual proof, the notion of *occurrences* must be introduced. Intuitively, the occurrences of an LI l are merely the Move nodes in the derivation tree that operated on one of l's licensee features. It does not

take much insight to realize that the first occurrence of l has to be some Move node that dominates it (otherwise l's licensee features could not be operated on) and is not included in slice(l) (no LI may license its own movement). One can even require the first occurrence to be the very first Move node satisfying these properties, thanks to the SMC (the reader might want to reflect on this for a moment). The reasoning is similar for all other occurrences, with the sole exception that closeness is now relativized to the previous occurrence. In more formal terms: Given an LI $l := \sigma :: \gamma c \delta$ with $c \in$ Base and $\delta := -f_1, \ldots, -f_n$, its occurrences occ_i, $1 \leq i \leq n$, are such that

- occ_1 is the first node labeled *move* that matches $-f_1$ and properly dominates the slice root of l.
- occ_i is the first node labeled *move* that matches $-f_i$ and properly dominates occ_{i-1}.

Note that every well-formed MDTL obeys the following two conditions:

- *M1*: For every LI l with $1 \leq n \leq |\delta|$ licensee features, there exist nodes $m_1, \ldots, m_n$ labeled *move* such that m_i is the i^{th} occurrence of l, $1 \leq i \leq n$.
- *M2*: For every node m labeled *move*, there is exactly one LI l such that m is an occurrence of l.

In fact, the implication holds in both directions.

Lemma 1. *For every MG G it holds that if $t \in T_{Lex_G \cup \{merge, move\}}$ is a combination of well-formed slices and respects all constraints on the distribution of Merge nodes, then it is well-formed iff M1 and M2 are satisfied.*

Proof. As just discussed the left-to-right direction poses little challenge. In the other direction, I show that μ^{-1} is well-defined on t and maps it to a well-formed $s \in \text{sder}(G)$. For LIs and Merge nodes, μ^{-1} is well-defined by assumption if it is well-defined for Move nodes. From the definition of *move* and the SMC it follows that $\mu^{-1}(move(\natural, t_2))$ (the expression returned by *move* when applied to $\mu^{-1}(t_2)$) is well-defined only if the root of t_2 is an expression consisting of at least two chains such that 1) its first chain has some feature $+f$ as its first feature and 2) the feature component of exactly one chain begins with $-f$. However, the former follows from the well-formedness of slices, while the latter is enforced by M2; in particular, if the SMC were violated, some Move node would be an occurrence for more than one LI. This establishes that μ^{-1} is well-defined for all nodes. Now $\mu^{-1}(t)$ can be ungrammatical only if the label of the root node contains some licensor or licensee features. The former is ruled out by M2 and the initial assumption that all slices are well-formed, whence every licensor feature is instantiated by a Move node. In the latter case, there must be some licensee feature without an occurrence in t, which is blocked by M1. □

Now we can finally move on to the proof of Thm. 5.

Proof. Given some k-local MG G with $L := \text{mder}(G) \in \text{MDTL}[merge, move^{(k)}]$, let κ-*factors*(L) be the set containing all κ-factors of L, and F the corresponding

strictly κ-local language built from these κ-factors. It is obvious that $F \supseteq L$, so one only needs to show that $F \subseteq L$. Trivially, $t \in L$ iff $t \in F$ for all trees t of depth $d \leq \kappa$. For this reason, only trees of size greater than κ will be considered.

Assume towards a contradiction that $F \not\subseteq L$, i.e. there is a t such that $F \ni t \notin L$. Clearly $F \ni t \notin L$ iff some condition enforced by *merge* or *move* on the combination of slices is violated, as the general restrictions on tree geometry (distribution of labels, length and directionality of slices) are always satisfied by virtue of κ always exceeding $|\gamma| + 1$. I now consider all possible cases. In each case, I use the fact that the constraints imposed by *merge* and *move* operate over a domain of bounded size less than κ, so that if $t \in F$ violated one of them, one of its κ-factors would have to exhibit this violation, which is impossible as κ-*factors*$(F) = \kappa$-*factors*(L).

Case 1 [Merge]: *merge* is illicit only if there is an internal node n labeled *merge* in slice(l) such that the shortest path from n to l is of length $1 \leq i$, the i^{th} feature of l is $+f$ for some $f \in$ BASE, and there is no LI l' such that the left daughter of n belongs to slice(l') and l' carries feature f. But the size of every slice is at most $|\gamma| + 1$, so the distance between n and l is at most $|\gamma|$, and that between n and l' at most $|\gamma| + 1$. Hence a factor of size $|\gamma| + 2$ is sufficient, which is less than κ. So if we found such a configuration, it would be part of some $\kappa_i \in \kappa$-*factors*$(F) = \kappa$-*factors*(L). Contradiction.

Case 2 [Move]: Conditions M1 and M2 can be split into three subcases.

Case 2.1 [Too few occurrences]: Assume that LI l has $j \leq |\delta|$ licensee features but only $i < j$ occurrences. Since $L \in$ MDTL$[merge, move^{(k)}]$, the shortest path spanning from any LI to its last occurrence includes nodes from at most $|\delta| * k + 1$ distinct slices. Since the size of no slice minus its LI exceeds $|\gamma|$, some factor κ_i of size greater than $(|\gamma| * |\delta| * k) + (|\gamma| + 1) \leq \kappa$ must exhibit the illicit configuration, yet $\kappa_i \notin \kappa$-*factors*(L).

Case 2.2 [Too many Move nodes]: Assume that for some Move node m there is no LI l such that m is an occurrence of l. This is simply the reverse of Case 2.1, where we obtain a violation if it holds for no LI l in any κ_i that m is one of its occurrences. But then at least one of these κ_i cannot be in κ-*factors*(L).

Case 2.3 [SMC violation]: The SMC is violated whenever there are two distinct items l and l' for which Move node m is an occurrence. As 2.2, this is just a special case of 2.1. □

We now have a very good approximation of MDTL$[merge, move^{(k)}]$ for any choice of $k > 0$. They are not local, or equivalently, strictly 2-locally testable, but they are strictly κ-locally testable, where κ depends on k and the maxima of licensor and licensee features, respectively. But what about MDTL$[merge, move]$ in general?

To readers acquainted with MGs it will hardly be surprising that unrestricted MDTLs are not strictly locally testable. Nor is it particularly difficult to demonstrate that they even fail to be locally threshold testable. In [1], it was proved that closure under k-guarded swaps is a necessary condition for a language to be definable in $\text{FO}_{mod}[S_1, S_2]$ — that is to say, first-order logic with unary predicates for all labels, binary predicates for the left child and right child relations, respec-

tively, and the ability to perform modulo counting. Evidently $\mathrm{FO}_{mod}[S_1, S_2]$ is a proper extension of $\mathrm{FO}[S_1, S_2]$, and definability in the latter fully characterizes the locally threshold testable languages [20]. So no language that isn't closed under k-guarded swaps is locally threshold testable.

Definition 8. *Let $t := C \cdot \Delta_1 \cdot \Delta \cdot \Delta_2 \cdot T$ be the composition of trees $C := C_t[a, x)$, $\Delta_1 := C_t[x, y)$, $\Delta := C_t[y, x')$, $\Delta_2 := C_t[x', y')$ and $T := t|_{y'}$. The* vertical swap *of t between $[x, y)$ and $[x', y')$ is the tree $t' := C \cdot \Delta_2 \cdot \Delta \cdot \Delta_1 \cdot T$. If the subtrees rooted at x and x' are identical up to and including depth k, and the same holds for the subtrees rooted at y and y', then the vertical swap is* k-guarded.

Theorem 6. MDTL$[merge, move]$ *and the class of tree languages definable in* $\mathrm{FO}_{mod}[S_1, S_2]$ *are incomparable.*

Proof. Consider a grammar containing (at least) the following four items:

$$\mathrm{a} :: \mathrm{a} \quad \mathrm{a} :: \mathrm{a} - \mathrm{b} \quad \mathrm{a} :: = \mathrm{a}\ \mathrm{a} \quad \mathrm{a} :: = \mathrm{a} + \mathrm{b}\ \mathrm{a}$$

I restrict my attention to those derivation trees in which movement occurs exactly once. Pick any $k \in \mathbb{N}$. Then there is some derivation tree that can be factored as above such that Δ_1 contains the movement node at some depth $m > k$, Δ_2 contains the corresponding LI $\mathrm{a} :: \mathrm{a} - \mathrm{b}$ at some depth $n > k$, $C = \Delta = T$, and the depth of Δ and T exceeds k. Given this configuration, the vertical swap of Δ_1 and Δ_2 is k-guarded, yet $t' := C \cdot \Delta_2 \cdot \Delta \cdot \Delta_1 \cdot T$ is not a Minimalist derivation tree, as the movement node no longer dominates $\mathrm{a} :: \mathrm{a} - \mathrm{b}$, thereby negating closure under k-guarded swaps. □

The insufficiency of $\mathrm{FO}_{mod}[S_1, S_2]$ puts a strong lower bound on the complexity of MDTL$[merge, move]$. In the next section, I show that enriching $\mathrm{FO}[S_1, S_2]$ with proper dominance and equivalence is all it takes to make MDTL$[merge, move]$ first-order definable.

4 Definability in First-Order Logic

I start with an $\mathrm{FO}[S_1, S_2]$ theory of MDTL$[merge]$, which is then extended to $\mathrm{FO}[S_1, S_2, <, \approx]$ for MDTL$[merge, move]$. Given an MG G, $\mathrm{FO}[S_1, S_2]$ is defined over ordered binary branching trees in the standard way, with the signature containing a unary predicate p for each $p \in \Lambda := Lex_G \cup Op_G \cup \{\natural\}$ and binary predicates S_1 and S_2 for the left and right child relation, respectively. The equivalence relation is superfluous for MDTL$[merge]$. I write $x \lhd_1 y$ instead of $S_1(x, y)$, and similarly for S_2. Moreover, $x \lhd y$ iff $x \lhd_1 y \vee x \lhd_2 y$.

First a number of constraints are established to ensure that every node has exactly one label drawn from Λ, and that the arity of the labels is respected (it suffices only to restrict nullary symbols to leaves, as this entails that binary symbols can be assigned only to interior nodes). Furthermore, $\natural$ may be assigned to a node if and only if it is the left daughter of a Move node.

$$\forall x \left[\Big(\bigvee_{u \in \Lambda} u(x) \Big) \wedge \bigwedge_{u \in \Lambda} \Big(u(x) \rightarrow \bigwedge_{v \in \Lambda \setminus \{u\}} \neg v(x) \Big) \right]$$

$$\forall x\Big[\bigvee_{u\in\Lambda\setminus\{merge,move\}} u(x) \leftrightarrow \neg\exists y[x \lhd y]\Big]$$

$$\forall x\forall y\big[\natural(y) \leftrightarrow move(x) \wedge x \lhd_1 y\big]$$

As was pointed out in Sec. 2, MDTLs can be viewed as the result of combining the slices defined by LIs in all possible ways such that the constraints of the feature calculus are respected. Hence I first define the shape of slices before moving on to the feature conditions enforced by Merge. To simplify this task, I use $\searrow^n \phi(x)$ as a shorthand for "ϕ holds at the node reached from x by taking n steps down the right branch". The analogous $\swarrow^n \phi(x)$ moves us down the left branch instead, while $\nwarrow^n \phi(x)$ moves us upwards only along a right branch. Intuitively, $\searrow$, $\swarrow$ and $\nwarrow$ can be viewed as first-order implementations of modal diamond operators.

$$\searrow^0 \phi(x) \leftrightarrow \phi(x)$$

$$\searrow^n \phi(x) \leftrightarrow \exists y[x \lhd_2 y \wedge \searrow^{n-1} \phi(y)]$$

Recall that all slices are strictly right-branching and never exceed size $|\gamma|+1$. This is equivalent to saying that there is no node that is at least $|\gamma|+1$ S_2-steps away from a node satisfying a tautology $\top$.

$$\neg\exists x[\searrow^{|\gamma|+1} \top(x)]$$

Next, every interior node n must be licensed by a feature of the LI of the slice containing n. Again a special notational device proves useful: for any feature f, $f_i(x)$ holds iff for some $l \in Lex_G$ whose i^{th} feature is f, $l(x)$ is true (the index will be suppressed whenever the position of the feature is irrelevant). Now let $slr_i(x) \leftrightarrow \bigvee_{f\in\textsc{Base}} = f_i(x)$ and $lcr_i(x) \leftrightarrow \bigvee_{f\in\textsc{Base}} + f_i(x)$.

$$\forall x\Big[\Big(merge(x) \rightarrow \bigvee_{1\le i\le|\gamma|} \searrow^i slr_i(x)\Big) \wedge \Big(move(x) \rightarrow \bigvee_{1\le i\le|\gamma|} \searrow^i lcr_i(x)\Big)\Big]$$

Besides the evident restriction on the distribution of *merge* and *move*, the formula above also ensures that no $l \in \Lambda$ without selector or licensor features can ever be a right leaf.

We still have to establish a minimum size on slices, though, which is easily accomplished by requiring every selector/licensor feature to license a unique interior node.

$$\forall x\Big[\bigwedge_{1\le i\le|\gamma|} \Big(\big(slr_i(x) \rightarrow \nwarrow^i merge(x)\big) \wedge \big(lcr_i(x) \rightarrow \nwarrow^i move(x)\big)\Big)\Big]$$

Note that this also prevents every LI with selector or licensor features from occurring on a left branch. The topmost slice in the derivation is also subject to the condition that the category of its LI must be final.

$$\forall x\Big[\neg\exists y[y \lhd x] \rightarrow \bigvee_{\substack{c\in F\\ 0\le i\le|\gamma|}} \searrow^i c(x)\Big]$$

So far, then, our first-order theory enforces the correct minimum/maximum size of slices for every $l \in Lex_G$ and fixes their branching direction and node labels. For MDTL[*merge*], it only remains to capture the feature dependencies imposed by Merge: the category feature of the LI of the slice on the left branch has to match the selector feature of the LI found along the right branch.

$$\forall x \Big[merge(x) \rightarrow \bigwedge_{c \in \textsc{Base}} \Big(\swarrow^1 \bigvee_{0 \leq i \leq |\gamma|} \searrow^i c(x) \leftrightarrow \bigvee_{1 \leq j \leq |\gamma|} \searrow^j = c_j(x)) \Big) \Big]$$

Extending this basis to unrestricted MDTLs is surprisingly easy using the notion of occurrences we encountered earlier on. First, proper dominance and equivalence are added to the signature of FO[S_1, S_2], yielding FO[$S_1, S_2, <, \approx$]. As before, I use infix notation for all binary relations, so instead of $<(x, y)$ I write $x \lhd^+ y$. For every $i \leq |\delta|$, $match_i(x, y)$ denotes that x is associated to a feature that matches the i^{th} licensee feature of y.

$$\begin{aligned} &match_i(x, y) \leftrightarrow \\ &\quad \bigvee_{f \in \textsc{Base}} \Big(\bigwedge_{\substack{c \in \textsc{Base} \\ 1 \leq j \leq |\gamma|+1}} \big(c_j(y) \rightarrow -f_{j+i}(y)\big) \wedge move(x) \wedge \bigvee_{1 \leq g \leq |\gamma|} \searrow^g + f_g(x) \Big) \end{aligned}$$

Furthermore, the predicate $x \blacktriangleleft y \leftrightarrow \exists z, \exists z' \big[(x \lhd^+ z \vee x \approx z) \wedge z \lhd_1 z' \wedge (z \lhd^+ y \vee z \approx y)\big]$ holds of x and y iff x properly dominates y and they belong to different slices. Building on these two notions, it is a straightforward task to recast the definition of occurrences in first-order terms.

$$occ_1(x, l) \leftrightarrow match_1(x, l) \wedge x \blacktriangleleft l \wedge \neg \exists y \Big[x \lhd^+ y \wedge match_1(y, l) \wedge y \blacktriangleleft l \Big]$$

$$\begin{aligned} occ_i(x, l) \leftrightarrow x \lhd^+ l \wedge match_i(x, l) \wedge \exists y \Big[&x \lhd^+ y \wedge occ_{i-1}(y, l) \wedge \\ &\neg \exists z \big[x \lhd^+ z \wedge z \lhd^+ y \wedge match_i(z, l) \big] \Big] \end{aligned}$$

In line with Lem. 1, constraining the distribution of *move* requires but three formulas that demand, respectively, that every licensee feature has a matching move node, that every mode node has a matching licensee feature, and that no movement node can be matched against more than one licensee feature (SMC). It is only this very last condition that depends on the equivalence predicate as there is no other first-order definable way of distinguishing nodes (the use of equivalence in the definition of $\blacktriangleleft$ is merely a matter of convenience and can easily be avoided).

$$\forall x \Big[\bigwedge_{\substack{c \in \textsc{Base} \\ 1 \leq i \leq |\gamma|+1}} \Big(c_i(x) \rightarrow \bigwedge_{\substack{f \in \textsc{Base} \\ 0 \leq j \leq |\delta|}} \big(-f_{i+j}(x) \rightarrow \exists y \big[occ_j(y, x) \big] \big) \Big) \Big]$$

$$\forall x\Big[move(x) \rightarrow \exists l\big[\bigvee_{1\leq i\leq|\delta|} occ_i(x,l)\big]\Big]$$

$$\forall x\forall l\Big[\bigwedge_{1\leq i\leq|\delta|}\Big(occ_i(x,l) \rightarrow \forall l'\Big[\bigwedge_{j\in[|\delta|]\setminus\{0,i\}} \neg occ_j(x,l')\wedge\big(occ_i(x,l') \rightarrow l \approx l'\big)\Big]\Big)\Big]$$

Conclusion

The results reported herein highlight the rather indirect relation between MDTLs and the string languages they derive. MGs without movement yield context-free string languages, whereas even bounded movement is sufficient to generate all multiple context-free languages. At the level of tree languages, however, both movement-free and k-local MGs are strictly locally testable, whereas unrestricted movement leads to an increase in complexity that pushes MDTLs out of the realm of local threshold testability (see Fig. 2 on the next page).[1]

As my results posit a split between k-local and unrestricted MGs on the level of derivation trees, they seem to vindicate the assumption commonly made by syntacticians that locality restrictions on movement are a fundamental property of natural language that keeps computational complexity in check. On the other hand, weak generative capacity remains unaffected, and the locality rank is immaterial, as all local grammars can be made 1-local. Further work is needed before a full understanding can be reached as to how derivational complexity may interact with string language complexity, what measure of complexity should be used, and how this relates to syntactic proposals.

It must also be pointed out that alternative representations of Minimalist derivation trees could conceivably paint a different picture. Eventually, one would like to have a better understanding as to which aspects of a derivation tree language genuinely reflect the complexity of the derivational machinery underlying the MG formalism and which are just notational quirks. By probing different formats for Minimalist derivation trees we might also unearth new connections

[1] The strictly local nature of movement-free and k-local MDTLs also implies that they can be recognized by deterministic tree-walking automata. I conjecture that this does not carry over to unrestricted MDTLs unless the automata are enriched with two weak pebbles. In particular, non-deterministic tree-walking automata cannot recognize unrestricted MDTLs: The fundamental problem one faces while sifting through a derivation tree with unrestricted movement in a sequential manner is that either 1) the automaton has to keep track of an unbounded number of features when performing a brute-force search for an LI matching a given movement node, or 2) it gets lost in the derivation tree and cannot make its way back to the movement node in question. This makes it impossible to ensure that every movement node is an occurrence for exactly one LI, and non-determinism offers no remedy. The addition of two pebbles, on the other hand, allows the automaton to mark the movement node and the LI that was inspected last, so that the automaton can always find its way back and can infer from the position of the second pebble which LIs have already been looked at.

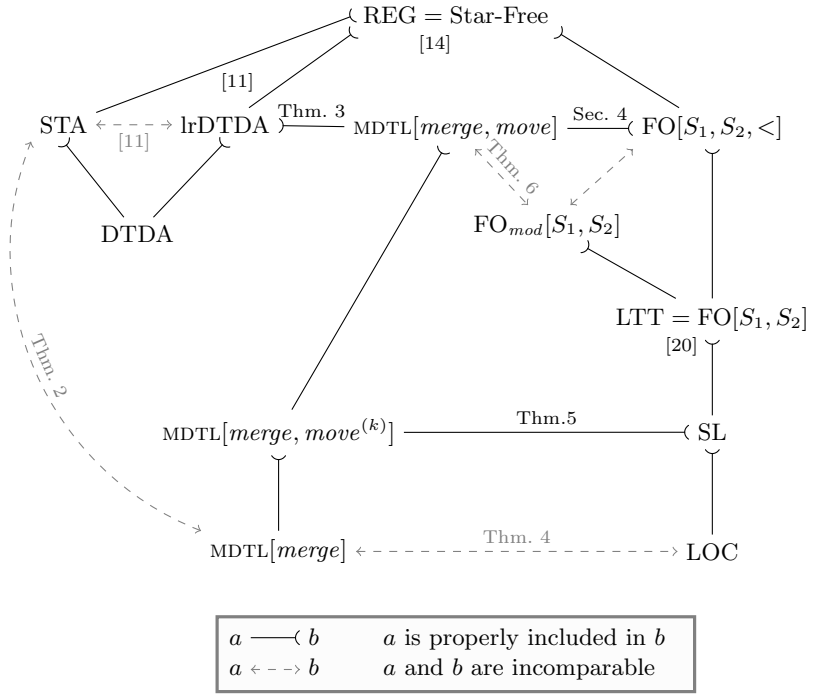

Fig. 2. MDTLs in the subregular space of strictly binary branching tree languages (references omitted for obvious relations)

between MGs and Tree Adjoining Grammar, an area that has recently enjoyed increased interest.

Acknowledgments My thanks go to Ed Stabler and the three anonymous reviewers for their helpful criticism. The research reported herein was supported by a DOC-fellowship of the Austrian Academy of Sciences.

Bibliography

[1] Benedikt, M., Segoufin, L.: Regular tree languages definable in FO and in FOmod. ACM Transactions in Computational Logic 11, 1–32 (2009)
[2] Chomsky, N.: The Minimalist Program. MIT Press, Cambridge, Mass. (1995)
[3] den Dikken, M.: Arguments for successive-cyclic movement through SpecCP. A critical review. Linguistic Variation Yearbook 9, 89–126 (2009)
[4] Graf, T.: Closure properties of minimalist derivation tree languages. In: Pogodalla, S., Prost, J.P. (eds.) LACL 2011. LNAI, vol. 6736, pp. 96–111 (2011)

[5] Gécseg, F., Steinby, M.: Tree Automata. Academei Kaido, Budapest (1984)
[6] Harkema, H.: A characterization of minimalist languages. In: de Groote, P., Morrill, G., Retoré, C. (eds.) Logical Aspects of Computational Linguistics (LACL'01), LNAI, vol. 2099, pp. 193–211. Springer, Berlin (2001)
[7] Kobele, G.M.: Minimalist tree languages are closed under intersection with recognizable tree languages (2011), to appear in Proceedings of LACL2011
[8] Kobele, G.M., Retoré, C., Salvati, S.: An automata-theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds.) Model Theoretic Syntax at 10. pp. 71–80 (2007)
[9] Mainguy, T.: A probabilistic top-down parser for Minimalist grammars (2010), arXiv:1010.1826v1
[10] Martens, W.: Static Analysis of XML Transformation- and Schema Languages. Ph.D. thesis, Hasselt University (2006)
[11] Martens, W., Neven, F., Schwentick, T.: Deterministic top-down tree automata: Past, present, and future. In: Proceedings of Logic and Automata 2008. pp. 505–530 (2008)
[12] Michaelis, J.: Transforming linear context-free rewriting systems into minimalist grammars. LNAI 2099, 228–244 (2001)
[13] Nivat, M., Podelski, A.: Minimal ascending and descending tree automata. SIAM Journal on Computing 26, 39–58 (1997)
[14] Potthoff, A., Thomas, W.: Regular tree languages without unary symbols are star-free. In: Proceedings of the 9th International Symposium on Fundamentals of Computation Theory. pp. 396–405 (1993)
[15] Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)
[16] Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) Logical Aspects of Computational Linguistics, LNCS, vol. 1328, pp. 68–95. Springer, Berlin (1997)
[17] Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–643. Oxford University Press, Oxford (2011)
[18] Stabler, E.P.: Top-down recognizers for MCFGs and MGs. In: Proceedings of the 2011 Workshop on Cognitive Modeling and Computational Linguistics (2011), to appear
[19] Stabler, E.P., Keenan, E.: Structural similarity. Theoretical Computer Science 293, 345–363 (2003)
[20] Thomas, W.: Languages, automata and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, New York (1997)
[21] Verdú-Mas, J.L., Carrasco, R.C., Calera-Rubio, J.: Parsing with probabilistic strictly locally testable tree languages. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 1040–1050 (2005)