A Single Movement Normal Form for Minimalist Grammars

Thomas Graf(✉), Alëna Aksënova, and Aniello De Santo

Department of Linguistics, Stony Brook University, Stony Brook, USA
mail@thomasgraf.net

Abstract. Movement is the locus of power in Minimalist grammars (MGs) but also their primary source of complexity. In order to simplify future analysis of the formalism, we prove that every MG can be converted into a strongly equivalent MG where every phrase moves at most once. The translation procedure is implemented via a deterministic linear tree transduction on the derivation tree language and induces at most a linear blow-up in the size of the lexicon.

Keywords: Minimalist grammars · Linear tree transductions · Derivation trees · Lexical blow-up · Successive cyclic movement

Introduction

Minimalist grammars (MGs; [17,18]) can be viewed as an extension of context-free grammars where the left-to-right order of leaves in the derivation tree does not necessarily correspond to their linear order in the string yield. These differences in the string yield are a side-effect of the operation *Move*, which removes a subtree from the derivation tree and reinserts it in a different position. Standard MGs are defined in such a way that one and the same subtree may be moved several times. In this case, the subtree is inserted only in the position determined by the final movement step, all previous steps have no tangible effect. This intuitive sketch suggests that these non-final—also called *intermediate*—movement steps can be omitted without altering the generated tree and string languages, a fact we prove in this paper.

The vacuity of intermediate movement is already implicit in the MCFG-equivalence proofs of [9,15]. We improve on this with a fully explicit translation in terms of a deterministic linear tree transduction over Minimalist derivation trees that yields MGs in *single movement normal form* (SMNF), i.e. MGs where every lexical item moves at most once (Sect. 2 and Appendix A). By skipping MCFGs as an intermediate step, the translation should prove easier to generalize to non-standard MGs for which no MCFG translation has been worked out in the literature. We also study the effects of SMNF on grammar size and demonstrate that the induced blow-up is highly dependent on movement configurations, but at most linear (Sect. 3.1). We furthermore discuss possible applications of SMNF (Sect. 3.2)—including certain parallels between syntax and phonology—and we

© Springer-Verlag Berlin Heidelberg 2016
A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 200–215, 2016.
DOI: 10.1007/978-3-662-53042-9_12

explore the ramifications of our result for the Chomskyan tradition of Minimalist syntax, which MGs are modeled after (Sect. 3.3).

1 Defining Minimalist Grammars

Due to space constraints we presume that the reader is already familiar with MGs in general [17,18], and their constraint-based definition in particular [5,7]. Following [14], we decompose MGs into a regular Minimalist derivation tree language (MDTL) and a mapping from derivation trees to phrase structure trees.

Definition 1. *Let* Base *be a non-empty, finite set of* feature names*. Furthermore,* Op $:= \{merge, move\}$ *and* Polarity $:= \{+, -\}$ *are the sets of* operations *and* polarities*, respectively. A* feature system *is a non-empty set* $Feat \subseteq$ Base $\times$ Op $\times$ Polarity*.*

Negative Merge features are called *category features*, positive Merge feature *selector features*, negative Move features *licensee features*, and positive Move features *licensor features*. A $(\Sigma, Feat)$-lexicon *Lex* is a finite subset of $\Sigma \times Feat^*$. Each member of *Lex* is a *lexical item* (LI) of the form $\gamma c \delta$, where γ is a string of licensor and selector features, c is a category feature, and δ is a string of 0 or more licensee features. A *Minimalist grammar* (MG) is $(\Sigma, Feat)$-lexicon coupled with a set $F \subseteq$ Base of *final categories*.

A ranked alphabet Σ is a finite union of finite sets $\Sigma^{(0)}, \ldots, \Sigma^{(n)}$ such that $\sigma \in \Sigma^{(i)}$ has *arity* i—we also write $\sigma^{(i)}$. For every such Σ, T_Σ is the language of finite Σ-trees recursively defined by (I) $\sigma^{(0)} \in T_\Sigma$, and (II) $\sigma(t_1, \ldots, t_n) \in T_\Sigma$ iff $\sigma \in \Sigma^{(n)}$ and $t_i \in T_\Sigma$, $1 \leq i \leq n$. A *free derivation tree over Lex* is a tree over alphabet $\left\{l^{(0)} \mid l \in Lex\right\} \cup \left\{\circ^{(1)}, \bullet^{(2)}\right\}$. An interior node m is *associated* to the i-th positive polarity feature of LI l iff it is the i-th mother of l (the i-th mother of l is the mother of its $(i-1)$-th mother, and l is its own 0-th mother). Such an m is the *slice root* of l iff the mother of m is not associated to l. A free derivation tree is *correctly projected* iff (I) every interior node is associated to the feature of exactly one LI, and (II) for every LI l with $\gamma := f_1 \ldots f_n$ the i-th mother of l exists and is labeled $\circ$ if f_i is a Move feature, and $\bullet$ otherwise.

A *Minimalist derivation tree* t is a correctly projected tree of some *Lex* that obeys four ancillary conditions. We first list the two constraints regulating Merge:

Merge. For every node m of t associated to selector feature f^+ of LI l, one of its daughters is the slice root of an LI l' with category feature f^-.

Final. If the root of t is the slice root of LI l, then the category feature of l is a final category.

Move is also subject to two constraints, which require ancillary terminology. Two features f and g *match* iff they differ only in their polarity. An interior node m matches a feature g iff the feature m is associated to matches g. For every free derivation tree t and LI l of t with string $\mathrm{f}_1^- \cdots \mathrm{f}_n^-$ of licensee features, $n \geq 0$, the *occurrences* of l in t are defined as follows:

– $occ_0(l)$ is the mother of the slice root of l in t (if it exists).
– $occ_i(l)$ is the unique node m of t labeled $\circ$ such that m matches $-f_i$, properly dominates occ_{i-1}, and there is no node n in t that matches $-f_i$, properly dominates occ_{i-1}, and is properly dominated by m.

The occurrence of l with the largest index is its *final occurrence*. For t and l as before, and every node m of t:

Move. There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i^{th} occurrence of l, $1 \leq i \leq n$.
SMC. If m is labeled $\circ$, there is exactly one LI for which m is an occurrence.

Given an MG G with lexicon *Lex*, the MDTL of G is the largest set of correctly projected derivation trees over *Lex* such that every tree in L satisfies the four constraints above. Note that each constraint defines a regular tree language, wherefore all MDTLs are regular [5,14,15].

MDTLs are easily mapped to phrase structure trees (see [7,14]). First we relinearize all siblings m and n such that m precedes n iff m is an LI with a selector feature or otherwise m is the slice root of some LI. Then we project phrases such that a given node is relabeled <only if its label is $\bullet$ and it is the mother of an LI with at least one selector feature. All other interior nodes are labeled>. The purpose of $<$ and $>$ is to indicate which branch (left or right, respectively) contains the head of the phrase. The result is a phrase structure tree where the first argument of a head is always linearized to its right, whereas all other arguments are linearized to the left.

For movement, assume l is an LI with at least one licensee feature and slice root r in the derivation tree. Add a branch from r to the final occurrence of l, and replace the subtree rooted by r with a trace. Every unary branching interior node furthermore receives a trace as a left daughter. At the very end, every LI $\sigma :: \mathrm{f_1 f_2 \cdots f_n}$ is replaced by σ to ensure that no features are present in the phrase structure trees. The combination of these steps can be carried out by a tree-to-tree transduction ϕ that is definable in monadic second-order logic. The tree language generated by MG G is the image of its MDTL under ϕ.

2 Single Movement Normal Form

This section establishes the core result of our paper: every MG G can be converted into a strongly equivalent MG that is in SMNF. Section 2.1 defines a linear tree transduction to rewrite MG derivation trees such that no LI moves more than once (a corresponding transducer is defined in the appendix). Section 2.2 then shows that these derivations still generate the same derived trees, which in combination with some auxiliary lemmata establishes the strong equivalence of standard MGs and MGs in SMNF.

2.1 A Linear Tree Transduction for Single Movement

In principle, single movement could mean that at most one movement step takes place in the whole derivation. But MGs satisfying such a strong constraint only generate context-free languages [cf. 12] and thus aren't even weakly equivalent to standard MGs. Instead, SMNF enforces the weaker condition that each LI undergoes at most one movement step. In other words, there is no intermediate movement; if an LI moves at all, it moves directly to its final landing site.

Definition 2 (SMNF). *An MG G with lexicon Lex is in* single movement normal form *iff every $l \in Lex$ has at most one licensee feature.*

The general idea for bringing an MG G into SMNF is very simple. Starting out with G's MDTL, one deletes from each derivation tree all those Move nodes that aren't a final occurrence for some LI. The removal of movement steps must be matched by (I) the removal of all non-final licensee features on the moving LIs and, (II) the removal of the licensor feature that each intermediate Move node was associated to.

It is this feature modification step that takes some finesse. Without further precautions, the resulting derivation may violate **SMC**, as is shown in Fig. 1 (here and in all following examples, Merge features are written in upper case and Move features in lower case). In the original derivation, LI b is the first to move, which is followed by two movement steps of LI a. Note that both undergo movement triggered by a licensee feature f^-. But the feature g^- triggering intermediate movement of a prevents f^- from becoming active on a until b has checked its feature f^-. Deleting g^- does away with this safeguard—both instances of f^- are active at the same time and receive the same occurrence, which is prohibited by **SMC**. The solution is to carefully rename features to avoid such conflicts while keeping the number of features finite.

All three steps—intermediate Move node deletion, intermediate feature deletion, and final feature renaming—can be carried out by a non-deterministic, linear bottom-up tree transducer τ, wherefore the regularity of the MDTL is preserved [3]. The definition of this transducer is rather unwieldy, so we opt for a high-level exposition at this point and relegate the detailed treatment to the appendix. Let l be an LI such that its final occurrence o is associated to feature

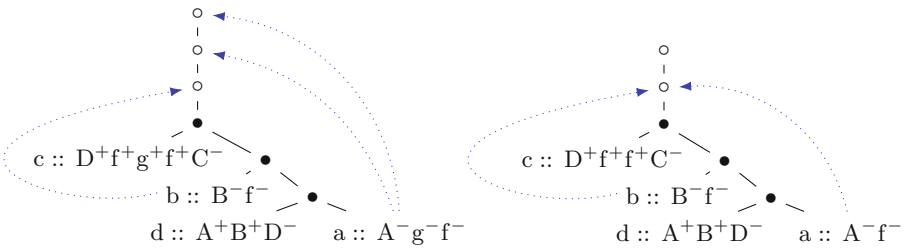

Fig. 1. Removal of intermediate movement steps may result in **SMC** violations

f and the path from o to l contains Move nodes $m_1, \ldots, m_n$ $(0 \leq n)$. Then the following steps are carried out in a bottom-up fashion:

- **Deletion**
  - remove all intermediate occurrences of l, and
  - for every removed intermediate occurrence, delete the licensor feature on the LI that said occurrence was associated to, and
- **Renaming**
  - replace l's string of licensee features by f_j^-, and
  - replace the feature on LI l' that o is associated to by f_j^+, where
  - $j \in \mathbb{N}$ is the smallest natural number distinct from every $k \in \mathbb{N}$ such that some m_i is associated to f_k^+, $1 \leq i \leq n$.

An instance of the mapping computed by τ is given in Fig. 2.

Subscripting is essential for the success of the translation. Intuitively, it may seem more pleasing to contract strings of licensee features into a single licensee feature, as suggested by one reviewer. So the LI $\sigma :: \gamma \mathrm{cf}_1^- \cdots \mathrm{f}_\mathrm{n}^-$ would become $\sigma :: \gamma \mathrm{c}[\mathrm{f}_1 \cdots \mathrm{f}_\mathrm{n}]^-$. But this does not avoid all **SMC** violations. Well-formed MG derivations can contain configurations where two LIs l and l' have the same string δ of licensee features, the final occurrence o of l dominates l', and the final occurrence of l' dominates o. In this case, atomizing δ into a single licensee feature δ^- would incorrectly make o a final occurrence of l', too.

It is also worth mentioning that even though the transducer τ as defined in the appendix is non-deterministic, the transduction itself is deterministic.

Lemma 1. *The transduction computed by τ is a function.*

Proof. Let t be some arbitrary Minimalist derivation tree and suppose that t has two distinct images t_1 and t_2 under τ. Inspection of τ reveals that t_1 and t_2 must be isomorphic and thus can only differ in the choice of indices for licensor and licensee features. We show by induction that these indices are always the same. Consider some arbitrary move node m of t_1 and t_2 that is associated to f_i^+ and f_j^+ in these derivations, respectively. If m does not properly dominate any other move nodes associated to some f_k^+, then $i = j = 0$. Otherwise, m properly dominates a sequence of move nodes $m_0, \ldots, m_n$ with each one associated to some subscripted version of f^+. By our induction hypothesis, each move node is associated to the same licensor feature f_k^+ in t_1 and t_2. But then there is a unique choice for the smallest natural number distinct from each one of these k, wherefore m is associated to the same licensor feature in t_1 and t_2. Consequently, $t_1 = t_2$ after all. □

The lemma shows that the non-determinism of τ is merely due to the restricted locality domain of linear transducers, which forces τ to make educated guesses about the shape of the derivation tree. A transducer with more elaborate lookup mechanism can compute the same transduction in a deterministic fashion. In particular, τ computes a deterministic, regularity preserving, first-order definable tree-to-tree transduction. In the remainder of this paper we take τ to directly refer to the transduction rather than any specific implementation thereof. We also write $\tau(t)$ for the image of t under τ.

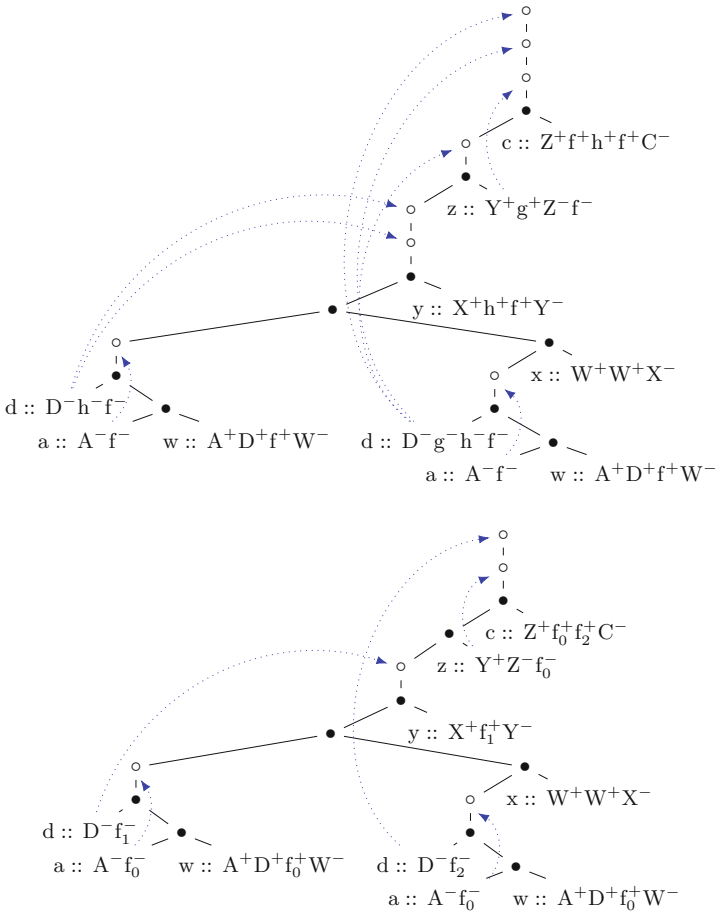

Fig. 2. Derivation tree (top) and its image under τ (bottom)

2.2 Proof of Strong Equivalence

We now show that the conversion carried out by τ does indeed result in an MG that generates the same set of phrase structure trees. To this end, we first establish an important restriction on movement. For every MG G, $\mu := |\{f \mid \langle f, move\rangle \times \textsc{Polarity} \in \mathit{Feat}\}|$. That is to say, μ indicates the number of distinct movement features. Given a derivation tree t of G and node n in t, the *traffic of* n, $\mathrm{traf}(n)$, is a measure of the number of LIs l in t that cross n during the derivation. More precisely, $\mathrm{traf}(n)$ denotes the number of LIs that are properly dominated by n in t and that have at least one occurrence that reflexively dominates n.

Lemma 2. *For every MG G, derivation tree t of G, and node n in t,* $0 \leq \mathrm{traf}(n) \leq \mu$.

Proof. The lower bound is trivial, while the upper bound follows from **SMC** and the definition of occurrence: there can be no two distinct LIs l and l' that are properly dominated by n and whose respective lowest occurrences reflexively dominating n are also associated to instances of the same licensor feature. Since there are only μ distinct licensor features, $\text{traf}(n)$ cannot exceed μ. □

Lemma 3. *It always holds for τ as defined above that $0 \leq j < \mu$.*

Proof. Consider an arbitrary Move node m. Since $\text{traf}(m) \leq \mu$, there cannot be more than μ LIs that are properly dominated by m and whose final (and only) occurrence reflexively dominates m. In order to distinguish the final occurrences of these LIs, then, one needs at most μ distinct indices. □

Lemma 3 guarantees both that τ is indeed a linear tree transduction and that the set of LIs occurring in the image of an MDTL under τ is still finite. As each one of these LIs has a well-formed feature string of the form $\gamma c \delta$, the set of LIs in the derivations produced by τ is an MG. It still remains to be shown, though, that the actual derivation trees produced by τ are well-formed.

Lemma 4. *It holds for every MDTL L and derivation tree $t \in L$ that $\tau(t)$ is a well-formed derivation tree.*

Proof. We have to show that $\tau(t)$ is correctly projected and satisfies **Merge**, **Final**, **Move** and **SMC**. The former holds because Move nodes are deleted iff their licensor features are removed. Furthermore, τ does not remove any Merge nodes or change any category or selector features, so **Merge** and **Final** hold because the domain of τ is an MDTL. Hence we only have to worry about **Move** and **SMC**.

Move. Move node m is an occurrence of some LI l in $\tau(t)$ (and thus its final occurrence) iff m is the lowest node in $\tau(t)$ that properly dominates l and is associated to a licensor feature matching l's single licensee feature. It is easy to see that τ furnishes at least one such m for every LI with a licensee feature.

SMC. We give an indirect proof that every Move node m is an occurrence for at most one LI in derivation tree t. This implies that m is an occurrence for exactly one LI thanks to **Move** and the fact that the images under τ contain as many Move nodes as LIs with licensee features.

Suppose that m in $\tau(t)$ is an occurrence for two distinct LIs l and l'. Then both l and l' have the same licensee feature f_i^-, for some arbitrary choice of f and i. Hence it must hold in t that

1. $\tau^{-1}(m)$ properly dominates $\tau^{-1}(l)$ and $\tau^{-1}(l')$, and
2. $\tau^{-1}(m)$ is a final occurrence o for either $\tau^{-1}(l)$ or $\tau^{-1}(l')$, and
3. $\tau^{-1}(m)$ is properly dominated by the final occurrence o' of the other.

Here $\tau^{-1}(n)$ denotes the node in t that corresponds to n. Now assume w.l.o.g. that o and o' are the final occurrences of $\tau^{-1}(l)$ and $\tau^{-1}(l')$, respectively. Then the path from $\tau^{-1}(l')$ to o' includes o, whence $\tau(o)$ and $\tau(o')$ must be associated to licensor features with distinct indices given the definition of τ. But then l and l' do not have the same licensee feature, either. Contradiction. □

We now know that τ produces a non-empty set of well-formed Minimalist derivation trees. This still does not imply, however, that the image of an MDTL under τ is itself an MDTL. The complicating factor is that MDTLs must be *maximal* sets of well-formed derivation trees—the LIs created by τ may allow for completely new derivations that are not part of the set produced by τ.

As a concrete example, consider the MG with the following lexicon:

$$\mathrm{u} :: \mathrm{B}^{+}\mathrm{C}^{-} \quad \mathrm{v} :: \mathrm{A}^{-}\mathrm{f}^{-}\mathrm{g}^{-} \quad \mathrm{w} :: \mathrm{B}^{+}\mathrm{g}^{+}\mathrm{B}^{-} \quad \mathrm{x} :: \mathrm{B}^{+}\mathrm{f}^{+}\mathrm{B}^{-} \quad \mathrm{y} :: \mathrm{A}^{+}\mathrm{B}^{-}$$

This grammar allows for only one derivation, depicted in Fig. 3 (left), which yields the string *uvwxy*. The image of this derivation tree under τ (see Fig. 3 right) contains almost exactly the same LIs. The only difference is the absence of f^{+} on x and f^{-} on v.

$$\mathrm{u} :: \mathrm{B}^{+}\mathrm{C}^{-} \quad \mathrm{v} :: \mathrm{A}^{-}\mathrm{g}^{-} \quad \mathrm{w} :: \mathrm{B}^{+}\mathrm{g}^{+}\mathrm{B}^{-} \quad \mathrm{x} :: \mathrm{B}^{+}\mathrm{B}^{-} \quad \mathrm{y} :: \mathrm{A}^{+}\mathrm{B}^{-}$$

Minor as this change may be, it has a major effect in that the MDTL is no longer singleton but actually infinite. Without the regulating effect of the licensee features of v, the LI x can be merged an arbitrary number of times. As a result, the generated string language is now $uvwx^{*}y$ instead of *uvwxy*.

Fortunately this issue is easy to avoid. It is always the case that the range R of τ is a subset of the unique MDTL over the set of LIs that occur in the derivation trees of R. Furthermore, R is guaranteed to be regular because MDTLs are regular and τ preserves regularity. Finally, the intersection of an MDTL and a regular tree language is the MDTL of some MG, *modulo* relabeling of selector and category features [4,7,13]. So to ensure that τ yields an MDTL, we only have to intersect its range with the MDTL over the set of all LIs that occur in at least one derivation tree produced by τ. A full algorithm for the intersection step is given in Sect. 3.2.3 of [7]. The algorithm operates directly on the MG lexicon, which allows it to be efficiently combined with the transducer in the appendix. The result is an MG G_{smnf} in SMNF.

It only remains for us to prove that G_{smnf} generates exactly the same derived trees as the original MG G, ignoring intermediate landing sites. To this end we first establish the weaker result that τ preserves the derived trees (again ignoring intermediate landing sites). Let ϕ be the standard mapping from derivation trees

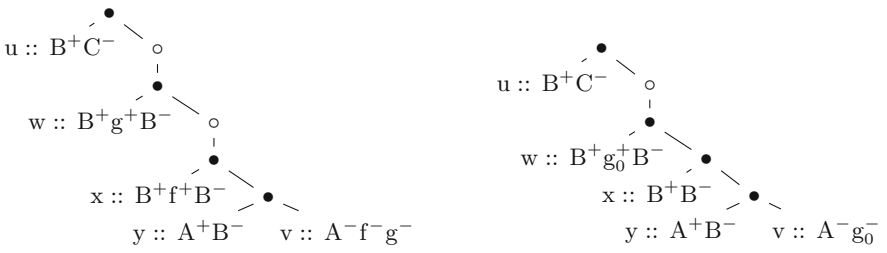

Fig. 3. Only possible derivation of example MG (left) and its image under τ (right)

to phrase structure trees and h a function that removes all intermediate landing sites from the phrase structure trees of an MG.

Theorem 1. *For every MG G with MDTL L, $h(\phi(L)) = \phi(\tau(L))$.*

Proof. Pick some arbitrary $t \in L$. Suppose $h(\phi(t)) \neq \phi(\tau(t))$. There must be some LI l with final occurrence o in t such that $\tau(o)$ is not the final occurrence of $\tau(l)$. But then either $\tau(l)$ has no final occurrence, the same final occurrence as some other LI, or the final occurrences of some LIs, including $\tau(l)$, have been switched. The former two are impossible as τ only generates well-formed derivation trees (Lemma 4). The latter is precluded by τ's choice of unique indices. Hence $h(\phi(t)) = \phi(\tau(t))$ after all and we have $h(\phi(L)) \subseteq \phi(\tau(L))$. The same arguments can be used in the other direction to show that for every $t \in \tau(L)$ and $t' \in \tau^{-1}(t)$ we have $h(\phi(t')) = \phi(t)$. This establishes $h(\phi(L)) \supseteq \phi(\tau(L))$, concluding the proof. □

Corollary 1. *For every MG G there is a strongly equivalent MG G_{smnf} in SMNF.*

Proof. The only addition to the previous proof is that G_{smnf} is obtained from the image of G's MDTL via the category refinement algorithm of [7]. Therefore G and G' may use different category and selector features. But since these features are all removed by ϕ, they are immaterial for the generated set of phrase structure trees. □

3 Evaluation

We now know that every MG has a strongly equivalent counterpart in SMNF. The relevance of such a normal form theorem, however, depends on the properties of the normal form and its overall usefulness. SMNF simplifies movement in MGs while preserving strong generative capacity, but does so at the expense of a larger number of movement features. As we discuss next in Sect. 3.1, the actual blow-up in the size of the lexicon is hard to predict because it depends on how strictly movers are tied to specific positions. The blow-up is still guaranteed to be at most linear, though. Sections 3.2 and 3.3 subsequently discuss applications and linguistic implications of SMNF.

3.1 Effects on Succinctness and Grammar Size

The conversion to SMNF has two sources of lexical blow-up. The first one is τ, which may replace a single LI of the form $\mathrm{a} :: \cdots \mathrm{h}^{+} \cdots \mathrm{A}^{-} \mathrm{g}^{-} \cdots \mathrm{f}^{-}$ by multiple variants $\mathrm{a} :: \cdots \mathrm{h}_{\mathrm{i}}^{+} \cdots \mathrm{A}^{-} \mathrm{f}_{\mathrm{j}}^{-}$ that only differ in the value of i and j ($0 \leq i, j < \mu$) and which licensor features have been removed. Given an MG G, the lexicon size of its SMNF counterpart is thus linearly bounded by $\sum_{l \in Lex} \mu^{\gamma(l)+\delta(l)}$, where $\gamma(l)$ is the number of licensor features of l, and $\delta(l)$ is 1 if l contains a licensee feature and 0 otherwise. The second blow-up is due to the regular intersection step that

turns the range of τ into an MDTL. This step is known to be linear in the size of the original lexicon and polynomial in the size of the smallest (non-deterministic) automaton that recognizes the regular tree language [7]. Crucially, though, the size increase induced by τ depends greatly on the shape of the grammar.

In the optimal case, τ does not cause any lexical blow-up and thus defines a bijection between lexical items.[1] Intuitively, this is the case whenever the position of an f-mover is fixed with respect to other f-movers, such as in the MG for $a^n b^n d^n$ below that uses massive remnant movement:

$$\begin{array}{lll} \varepsilon :: \mathrm{A}^-\mathrm{a}^-\mathrm{f}^- & \mathrm{a} :: \mathrm{D}^+\mathrm{a}^+\mathrm{f}^+\mathrm{A}^-\mathrm{a}^-\mathrm{f}^- & \varepsilon :: \mathrm{D}^+\mathrm{a}^+\mathrm{f}^+\mathrm{A}'^- \\ \varepsilon :: \mathrm{B}^-\mathrm{b}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{A}^+\mathrm{b}^+\mathrm{f}^+\mathrm{B}^-\mathrm{b}^-\mathrm{f}^- & \varepsilon :: \mathrm{A}'^+\mathrm{b}^+\mathrm{f}^+\mathrm{B}'^- \\ \varepsilon :: \mathrm{D}^-\mathrm{d}^-\mathrm{f}^- & \mathrm{d} :: \mathrm{B}^+\mathrm{d}^+\mathrm{f}^+\mathrm{D}^-\mathrm{d}^-\mathrm{f}^- & \varepsilon :: \mathrm{B}'^+\mathrm{d}^+\mathrm{f}^+\mathrm{C}^- \end{array}$$

After applying τ, this yields a notational variant of the standard MG for this language. Crucially, both grammars have exactly the same number of LIs.

$$\begin{array}{lll} \varepsilon :: \mathrm{A}^-\mathrm{f}_0^- & \mathrm{a} :: \mathrm{D}^+\mathrm{f}_0^+\mathrm{A}^-\mathrm{f}_0^- & \varepsilon :: \mathrm{D}^+\mathrm{f}_0^+\mathrm{A}'^- \\ \varepsilon :: \mathrm{B}^-\mathrm{f}_1^- & \mathrm{b} :: \mathrm{A}^+\mathrm{f}_1^+\mathrm{B}^-\mathrm{f}_1^- & \varepsilon :: \mathrm{A}'^+\mathrm{f}_1^+\mathrm{B}'^- \\ \varepsilon :: \mathrm{D}^-\mathrm{f}_2^- & \mathrm{d} :: \mathrm{B}^+\mathrm{f}_2^+\mathrm{D}^-\mathrm{f}_2^- & \varepsilon :: \mathrm{B}'^+\mathrm{f}_2^+\mathrm{C}^- \end{array}$$

Without this restriction to fixed relative positions, blow-up can occur even if the grammar does not allow for remnant movement and only generates a finite language. In the following grammar, the three lexical items a, b, and d can be selected by a in arbitrary order, and their target sites are similarly free in their relative ordering.

$$\begin{array}{lll} \varepsilon :: \mathrm{T}^+\mathrm{C}^- & \mathrm{a} :: \mathrm{M}^-\mathrm{a}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{a}^+\mathrm{f}^+\mathrm{C}^- \\ \varepsilon :: \mathrm{M}^+\mathrm{M}^+\mathrm{M}^+\mathrm{T}^- & \mathrm{b} :: \mathrm{M}^-\mathrm{b}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{b}^+\mathrm{f}^+\mathrm{C}^- \\ & \mathrm{d} :: \mathrm{M}^-\mathrm{d}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{d}^+\mathrm{f}^+\mathrm{C}^- \end{array}$$

Conversion into SMNF increases the size from 8 to 20 since each instance of f^- and f^+ must be replaced by f_i^- and f_i^+, $0 \leq i \leq 2$. The size increase of SMNF thus correlates with the number of combinatorial options furnished by the interaction of Merge and Move. Future work will hopefully be able to characterize this correlation in greater detail.

[1] Strictly speaking the optimal case is for τ to reduce the size of the lexicon. But as far as we can tell this only happens with needlessly redundant MGs such as the one below, where the SMNF lexicon contains only 5 instead of 7 entries.

$$\begin{array}{lll} \mathrm{c} :: \mathrm{M}^+\mathrm{C}^- & \mathrm{m} :: \mathrm{M}^-\mathrm{g}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{C}^+\mathrm{g}^+\mathrm{B}^- \\ \mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{M}^+\mathrm{C}^- & \mathrm{m} :: \mathrm{M}^-\mathrm{h}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{C}^+\mathrm{h}^+\mathrm{B}^- \\ \mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{C}^- & & \end{array}$$

A minor change to the grammar immediately undoes the size benefits of SMNF. All it takes is to replace $\mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{M}^+\mathrm{C}^-$ by $\mathrm{c} :: \mathrm{B}^+\mathrm{M}^+\mathrm{f}^+\mathrm{C}^-$. The SMNF lexicon then has 8 entries instead of 7 (1 for b, 2 for m and 5 for c).

3.2 Usefulness and Applications

A normal form can serve a multitude of purposes. Our main motivation for SMNF is to simplify proofs and rigorous analysis. The availability of intermediate movement in MGs creates special cases that are hardly ever insightful. For example, the top-down parser in [19] includes two movement rules, one for final and one for intermediate movement, yet the latter does nothing of interest except eliminate a pair of licensor and licensee features. Similarly, the proof in [12] that MGs cannot generate mildly context-sensitive string languages without remnant movement has to cover intermediate movement as a special case that does not add anything of value. Quite generally, intermediate movement is hardly ever relevant but frequently introduces complications that distract from the core ideas of proofs and theorems.

In fact, SMNF has already proven indispensable in ongoing projects. In the wake of our theorem, Graf and Heinz [8] show that SMNF reduces the complexity of MDTLs and renders them very similar to dependencies found in phonology. It has been known for quite a while that MDTLs are subregular [5], and it has been conjectured that segmental phonology is tier-based strictly local [10]. These two insights are combined by [8]: an MDTL is a tier-based strictly local tree language iff its grammar is in SMNF. This suggests that syntax and phonology are very much alike at a sufficient level of formal abstraction.

With other formalisms, normal forms have also been useful for parsing and the construction of automata. Chomsky Normal Form, for instance, is indispensable to guarantee cubic time complexity for CKY parsing of context-free grammars. Greibach Normal Form, on the other hand, simplifies the construction of equivalent, real-time pushdown automata for these grammars. At this point it remains an open question whether SMNF offers comparable advantages in these areas. On the one hand SMNF simplifies movement dependencies, on the other hand any tangible parsing benefits may be so minor that they are vastly outweighed by the lexical blow-up of the SMNF conversion. One conceivable scenario is that SMNF offers a parsing advantage only for those grammars where lexical blow-up is minimal due to movement being more restricted. It would be interesting to see whether this subclass is sufficiently powerful from a linguistic perspective. If so, it might indicate that natural languages restrict how movement dependencies interact in order to aid parsing.

3.3 Linguistic Implications

The unexpected parallels between phonology and MGs in SMNF, as well as our speculations above regarding parsing optimization, show that the existence of SMNF is not of purely technical interest but also raises deep linguistic questions. Yet there are certain linguistic concerns one might raise regarding SMNF.

In the syntactic literature, direct movement to the final landing site without intermediate movement steps is called *one fell swoop* movement. This kind of movement has been argued against on conceptual and empirical grounds (cf. Chap. 1 of [1]). However, the arguments against one fell swoop movement do

not carry over to SMNF. The reason for this is that the linguistic arguments are about derived trees, whereas SMNF is a property of derivation trees. In principle, nothing prevents us from modifying the mapping from derivation trees to derived trees so that landing sites are inserted where syntacticians want them to occur, e.g. at phase edges. Something along these lines was already proposed in [11, Sect. 2.1.1] for successive cyclic wh-movement. The strategy can easily be extended to other intermediate movement steps because most of them are readily predictable from the rest of the structure.

To take but one example, consider the movement steps of a subject wh-phrase as in *Who does John think kissed Mary?*, to which Minimalists would ascribe the structure $[_{\text{CP}}$ who C does John think $[_{\text{CP}}\ t$ C $[_{\text{TP}}\ t$ T t kissed Mary]]]. The subject starts out in Spec,VP of the embedded clause, moves to the embedded subject position in Spec,TP, then to the embedded Spec,CP, and from there finally to matrix Spec,CP. But all these intermediate movement steps are readily predictable from the fact that *who* moves to matrix Spec,CP. A phrase that starts out in Spec,VP and moves to a position above Spec,TP must land there by virtue of being a subject. A phrase that moves to some higher Spec,CP must land in every CP-specifier that occurs between the two. There is no requirement for intermediate movement steps to be linked to feature checking in the derivation because they can be inferred indirectly.

Cases where this indirect inference of intermediate movement steps is impossible are hard to come by. They mostly involve configurations where movement serves the sole purpose of modifying word order, such as scrambling or linearization of siblings to account for the head-initial/head-final contrast between languages. But it is far from evident that intermediate movement matters for scrambling, and the headedness parameter can be captured more directly by encoding the linearization of arguments in the selector features of MGs [cf. 6,18]. Given our current linguistic understanding, then, there is no sound argument or conclusive evidence against SMNF, a point that was already made over 25 years ago in [16, Sect. 3.4.1] (we are indebted to an anonymous reviewer for bringing this to our attention). The only major issue is the lexical blow-up, but this is as much a detriment as an opportunity: a hierarchy that ranks movement dependencies with respect to the SMNF blow-up they induce might furnish novel generalizations with rich typological implications.

Conclusion

Every MG can be efficiently transformed into a strongly equivalent MG in SMNF such that every LI moves at most once. The translation procedure in this paper is specified as a linear transduction over MDTLs, but is easily extended to a mapping between Minimalist lexicons: given an MG lexicon, one can immediately construct a deterministic bottom-up tree automaton that recognizes its MDTL [14], from which one obtains an automaton for the corresponding SMNF tree language via the usual transducer composition algorithm [2]. The nullary symbols of the automaton constitute the new lexicon. A Python implementation of this extended translation is hosted at https://github.com/CompLab-StonyBrook.

In future work, we hope to generalize SMNF from standard MGs to movement-generalized MGs [6]. We also intend to further explore how movement can be restricted to avoid lexical blow-up and whether these restrictions are linguistically feasible. It will also be interesting to see if some of these findings are applicable in the reverse direction to obtain algorithms that minimize (movement-generalized) MGs by adding intermediate movement steps.

A Specification of SMNF Transducer

A *bottom-up tree transducer* is a 5-tuple $\tau := \langle \Sigma, \Omega, Q, F, \Delta \rangle$, where Σ and Ω are ranked alphabets, Q is a finite set of states, $F \subseteq Q$ is the set of final states, and Δ is a finite set of *transduction rules*. Each transduction rule is of the form $f(q_1(x_1), \ldots, q_n(x_n)) \to q(t)$ such that f is an n-ary symbol in Σ ($n \geq 0$), $q, q_1, \ldots, q_n \in Q$, and t is a tree with node labels drawn from Ω and the nullary symbols $x_1, \ldots, x_n$. The transducer is *linear* iff each x_i may occur at most once in t. It is *non-deleting* iff each x_i occurs at least once in t. It is *non-deterministic* iff at least two transduction rules have the same left-hand side.

A $(\Sigma_1, \ldots, \Sigma_n)$-tree is a tree whose nodes are labeled with symbols from $\bigcup_{1 \leq i \leq n} \Sigma_i$. Given (Σ, Q)-tree u and (Ω, Q)-tree v, τ immediately derives v from u ($u \Rightarrow_\tau v$) iff there is a transduction rule such that u has the shape $f(q_1(u_1), \ldots, q_n(u_n))$—where each u_i is the subtree of u immediately dominated by q_i—and v is the result of substituting u_i for x_i in t. We use $\Rightarrow_\tau^+$ to denote the transitive closure of $\Rightarrow_\tau$. The transduction computed by τ is the set $\tau := \{\langle u, v \rangle \mid u \Rightarrow_\tau^+ q_f(v), u \text{ a } \Sigma\text{-tree, and } q_f \in F\}$. We furthermore let $\tau(s) := \{\langle s, t \rangle \in \tau\}$, and $\tau(L) := \bigcup_{s \in L} \tau(s)$ for L a tree language.

We now define a non-deterministic linear bottom-up tree transducer that brings Minimalist derivation trees into SMNF. The transducer is almost non-deleting as it only deletes intermediate Move nodes. Consequently, it can be regarded as the composition of a non-deterministic relabeling and a deterministic transducer that deletes Move nodes marked for removal. Before moving on, we introduce an additional piece of MG notation. In a standard MG, every useful LI must be of the form $\gamma c \delta$, where γ is a string of licensor and selector feature, c is a category feature, and δ is a string of 0 or more licensee features. Given a feature component s, $m(s)$ is obtained from s by removing all Merge features. We overload m such that for every LI $l := \sigma :: \text{s}$, $m(l) := m(s)$.

The SMNF transducer has to handle three tasks in parallel: (I) detect and delete intermediate Move nodes, (II) modify the feature components of LIs, and (III) ensure that each licensee feature is subscripted with the smallest possible natural number. Consequently, each state has a tripartite structure

$$\left\langle \begin{matrix} u_1, \ldots, u_n \\ m_1, \ldots, m_n \\ I_1, \ldots, I_n \end{matrix} \right\rangle$$

such that $n \leq \mu$ (the upper bound on the grammar's traffic), u_i keeps track of the unchecked Move features of some LI l, m_i records how $m(l)$ was modified, and I_i stores which required indices have not been encountered yet. More precisely:

for each u_i there is some LI l with u_i a suffix of $m(l)$; m_i is a string of indexed Move features and the distinguished symbol $\square$ such that removal of indices and $\square$ yields a subsequence of $m(l)$ including the final licensee feature; and I_i is some subset of the closed interval $[0, \mu-1]$ of natural numbers. Among all these states, the only final state is the empty state $\langle\rangle$.

While the transducer has a large number of rules, they can easily be compressed into a few templates using algebraic operations. First, we define a nondeterministic relabeling ℓ operating on MG feature strings that preserves all Merge features and either deletes Move features or relabels them:

$$\ell(f_1 \cdots f_n) := \begin{cases} f_1 \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a Merge feature} \\ f_{1,i} \ell(f_2 \cdots f_n) \text{ or } \square \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a licensor feature, } 0 \leq i < \mu \\ f_{1,i} \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a licensee feature, } 0 \leq i < \mu \\ \varepsilon & \text{if } f_1 \cdots f_n = \varepsilon \end{cases}$$

We extend ℓ to LIs: if $l := \sigma :: \gamma \mathrm{cf_1} \cdots \mathrm{f_n}$, then $\ell(l)$ is $\sigma :: \ell(\gamma \mathrm{c})$ if $n = 0$ and $\sigma :: \ell(\gamma \mathrm{cf_n})$ otherwise. In addition, h is a homomorphism that replaces the distinguished symbol $\square$ by ε in every string. The transduction rules for leaf nodes now follow a simple template:

$$\textbf{LIs}. \qquad \mathrm{l} :: \mathrm{s} \rightarrow \begin{cases} \left\langle \begin{matrix} m(l) \\ m(l') \\ \varepsilon \end{matrix} \right\rangle (h(l')) & \text{for } l' = \ell(l) \text{ if } l' \text{ does not end in a licensee feature} \\ \left\langle \begin{matrix} m(l) \\ m(l') \\ [0, k-1] \end{matrix} \right\rangle (h(l')) & \text{for } l' = \ell(l) \text{ if the licensee feature of } l' \text{ is subscripted with } k \end{cases}$$

For Merge we use a binary operator $\otimes$ that combines all the components of the states.

$$\left\langle \begin{matrix} u_1, \ldots, u_j \\ m_1, \ldots, m_j \\ I_1, \ldots, I_j \end{matrix} \right\rangle \otimes \left\langle \begin{matrix} u_{j+1}, \ldots, u_k \\ m_{j+1}, \ldots, m_k \\ I_{j+1}, \ldots, I_k \end{matrix} \right\rangle := \left\langle \begin{matrix} u_1, \ldots, u_j, u_{j+1}, \ldots, u_k \\ m_1, \ldots, m_j, m_{j+1}, \ldots, m_k \\ I_1, \ldots, I_j, I_{j+1}, \ldots, I_k \end{matrix} \right\rangle$$

$$\textbf{Merge}. \qquad \bullet(q(x), q'(y)) \Rightarrow q \otimes q'(\bullet(x, y))$$

The Move rules have to handle most of the work in the transducer. First, they have to delete movement features in the top component and use this information to decide whether the Move node is final or intermediate. Licensor features in the second component must also be removed, and the same goes for licensee features if the Move node is final. In the latter case, the index of the checked licensee feature is removed from all other index sets. Checking of a licensee feature, in turn, is only possible if its index set is empty.

As before, we simplify our presentation by using an algebraic operator $\ominus$, which takes care of updating index sets. Given a state q with index set I_j at position j, $I_j \ominus_f k = I_j - \{k\}$ if m_j ends in some subscripted version of f^-. In all other cases, $I_j \ominus_f k = I_j$. The transition rules for intermediate and final movement are now captured by four distinct cases. We only give two here, the other two are their mirror image with the order of $\mathrm{f}^+\delta_j$ and $\mathrm{f}^-\delta_k$ switched.

$$\textbf{Move}. \quad \circ \left(\left\langle \begin{matrix} u_1, \ldots, \mathrm{f}^+\delta_j, \ldots, \mathrm{f}^-\delta_k, \ldots, u_n \\ m_1, \ldots, m_j, \ldots, m_k, \ldots, m_n \\ I_1, \ldots, I_j, \ldots, I_k, \ldots, I_n \end{matrix} \right\rangle (x) \right)$$

$$:= \begin{cases} \left\langle \begin{matrix} u_1, \ldots, \delta_j, \ldots, \delta_k, \ldots, u_n \\ m_1, \ldots, m'_j, \ldots, m_k, \ldots, m_n \\ I_1, \ldots, I_j, \ldots, I_k, \ldots, I_n \end{matrix} \right\rangle (x) & \text{if } \delta_k \neq \varepsilon \text{ and } m_j = \square m'_j \\ \left\langle \begin{matrix} u_1, \ldots, \delta_j, \ldots, \ldots, u_n \\ m_1, \ldots, m'_j, \ldots, \ldots, m_n \\ I_1 \ominus_f i, \ldots, I_j, \ldots, \ldots, I_n \ominus_f i \end{matrix} \right\rangle (\circ(x)) & \text{if } \delta_k = \varepsilon,\ m_j = \mathrm{f}_\mathrm{i}^+ m'_j, \\ & m_k = \mathrm{f}_\mathrm{i}^-,\ I_k = \emptyset, \text{ and} \\ & \text{only } m_k \text{ starts with } \mathrm{f}_\mathrm{i}^- \end{cases}$$

Note that since the transducer is restricted to well-formed derivation trees, at most one component of a state can contain licensor features. Similarly, the SMC prevents any two u_i from starting with the same licensee feature, so the indices j and k in the template are always uniquely identified.

A few clarifying remarks may be in order. First, note that the transducer always halts if it finds a case of intermediate movement but did not delete the corresponding licensor feature earlier on. This is enforced by m_j starting with $\square$. Second, the index set I_j is not updated for final movement. That is because the corresponding LI has not started to move yet, so its index set is not active yet. If I_j were updated, then an LI that licenses f_2 movement would be allowed to undergo f_3 movement even if f_2 movement is possible, too.

To sum up, given an MG with lexicon Lex, the SMNF transducer τ has input alphabet $\Sigma := Lex^{(0)} \cup \{\circ^{(1)}, \bullet^{(2)}\}$ and output alphabet $\Omega := \bigcup_{l \in Lex} h(\ell(l))^{(0)} \cup \{\circ^{(1)}, \bullet^{(2)}\}$. Its state set Q consists of all possible tripartite tuples as defined at the beginning of this section. While this set is large, it is guaranteed to be finite. The empty state $\langle\rangle$ is the only final state, and the set Δ of transduction rules contains all possible instantiations of the templates above given Q.

References

1. Abels, K.: Successive cyclicity, anti-locality, and adposition stranding. Ph.D. thesis, University of Conneticut (2003)
2. Baker, B.S.: Composition of top-down and bottom-up tree transductions. Inf. Control **41**, 186–213 (1979)
3. Engelfriet, J.: Bottom-up and top-down tree transformations – a comparison. Math. Syst. Theor. **9**, 198–231 (1975)

4. Graf, T.: Closure properties of minimalist derivation tree languages. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 96–111. Springer, Heidelberg (2011)
5. Graf, T.: Locality and the complexity of minimalist derivation tree languages. In: Groote, P., Nederhof, M.-J. (eds.) Formal Grammar 2010/2011. LNCS, vol. 7395, pp. 208–227. Springer, Heidelberg (2012)
6. Graf, T.: Movement-generalized minimalist grammars. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 7351, pp. 58–73. Springer, Heidelberg (2012)
7. Graf, T.: Local and transderivational constraints in syntax and semantics. Ph.D. thesis, UCLA (2013)
8. Graf, T., Heinz, J.: Commonality in disparity: the computational view of syntax and phonology. In: Slides of a talk given at GLOW 2015, 18 April, Paris, France (2015)
9. Harkema, H.: A characterization of minimalist languages. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, pp. 193–211. Springer, Heidelberg (2001)
10. Heinz, J., Rawal, C., Tanner, H.G.: Tier-based strictly local constraints in phonology. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pp. 58–64 (2011)
11. Kobele, G.M.: Generating copies: an investigation into structural identity in language and grammar. Ph.D. thesis, UCLA (2006)
12. Kobele, G.M.: Without remnant movement, MGs are context-free. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10/11. LNCS, vol. 6149, pp. 160–173. Springer, Heidelberg (2010)
13. Kobele, G.M.: Minimalist tree languages are closed under intersection with recognizable tree languages. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 129–144. Springer, Heidelberg (2011)
14. 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)
15. Michaelis, J.: Transforming linear context-free rewriting systems into minimalist grammars. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, pp. 228–244. Springer, Heidelberg (2001)
16. Ristad, E.S.: Computational structure of human languages. Ph.D. thesis, MIT (1990)
17. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
18. Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–643. Oxford University Press, Oxford (2011)
19. Stabler, E.P.: Bayesian, minimalist, incremental syntactic analysis. Top. Cogn. Sci. **5**, 611–633 (2013)