Procesamiento del Lenguaje Natural, nº39 (2007), pp. 27-34 recibido 24-05-2007; aceptado 22-06-2007

A note on the complexity of the recognition problem for the Minimalist Grammars with unbounded scrambling and barriers*

Alexander Perekrestenko
Universidad Rovira i Virgili
Grupo de Investigación en Lingüística Matemática
(Research Group on Mathematical Linguistics)
International PhD School in Formal Languages and Applications
Pl. Imperial Tarraco 1, 43005 - Tarragona
alexander.perekrestenko@estudiants.urv.cat

Resumen: Las Gramáticas Minimalistas fueron introducidas recientemente como un modelo para la descripción formal de la sintaxis de los lenguajes naturales. En este artículo, se investiga una extensión no local de este formalismo que permitiría la descripción del desplazamiento optativo ilimitado de constituyentes sintácticos (*scrambling*), un fenómeno que existe en muchos idiomas y presenta dificultades para la descripción formal. Se establece que la extensión de las Gramáticas Minimalistas con scrambling sin la llamada condición del movimiento más corto (*shortest-move constraint*, SMC) y con barreras hace que el problema de reconocimiento para el formalismo resultante pertenezca a la clase NP-hard de la complejidad computacional.
Palabras clave: Sintaxis, análisis sintáctico, Gramáticas Minimalistas, orden de palabras, scrambling, complejidad computacional, lenguajes formales

Abstract: Minimalist Grammars were proposed recently as a model for the formal description of the natural-language syntax. This paper explores a nonlocal extension to this formalism that would make it possible to describe unbounded scrambling which is a discriptionally problematic syntactic phenomenon attested in many languages. It is shown that extending Minimalist Grammars with scrambling without shortest-move constraint (SMC) and with barriers makes the recognition problem for the resulting formalism NP-hard.
Keywords: Syntax, parsing, Minimalist Grammars, word order, scrambling, computational complexity, formal languages

1 *Introduction*

The formalization of the natural language syntax is important both from the theoretical and practical point view. It allows us to check the feasibility of the existing syntactic theories as models of how we process the language and provides a framework for creating practical applications—grammars and parsing systems. In the formalization of natural-language syntax, following classes of grammars usually come into consideration.

Right-liner (regular) grammars. These grammars can only be used for so-called *shallow parsing* since their capacity to assign structural descriptions to sentences is too limited.

Context-free grammars. While these grammars can describe a big part of the natural language syntax in the weak sense, they fail to assign appropriate structural descriptions to sentences containing discontinuous constituents.

Mildly context-sensitive formalisms. Mildly context-sensitive grammars (MCSG) were proposed as a mathematical model of the natural-language syntax that would be only as powerful as necessary for the correct description of the existing syntactic phenomena. The mildly context-sensitive formalisms best explored today are Tree-adjoining Grammars (TAGs) and Minimalist Grammars (MGs).

* This research work has been partially supported by the Russian Foundation for Humanities as a part of the project "The typology of free word order languages" (grant RGNF 06-04-00203a). The author would also like to express his utmost gratitude to the head of the Research Group on Mathematical Linguistics of the Rovira i Virgili University prof. Carlos Martín Vide for his encouragement and advice.

ISSN: 1135-5948

Computationally unrestricted formalisms. Unification-based syntactic theories with unrestricted structure sharing, such as Head-driven Phrase Structure Grammar (HPSG), strictly speaking do not belong to the class of restricted grammars since they are based on unification formalisms which are Turing-equivalent. The problem of the computational universality of the formalism itself is here solved with the design of grammars that do not exploit the full power of the formalism.

Whatever the grammar or the class of formalisms, it is crucially important for it to allow parsing in deterministic polynomial time basing on the length of the input, for otherwise its hight computational complexity (or incomputability) would disqualify it both as a feasible mathematical model of the human language competence and as a technically applicable framework.

2 Linguistic data

One of the most problematic phenomena for the formalization of the natural-language syntax is so-called *scrambling*, which is a non-obligatory reordering of syntactic constituents. Originally, the term *scrambling* was used to denote the argument permutation observed in the so-called *middlefield* (Mittelfelt) in German. This phenomenon occurs in many other languages as well, for example, in Japanese, Russian, Turkish, etc. The descriptionally most problematic class of this phenomenon is the so-called *unbounded scrambling* where the permutating arguments belong to different verbal heads. In this kind of scrambling, a linear reordering of the arguments leads to their displacement from the embedded infinitival clauses into the matrix clause. Since in theory there is no limit on the depth of the infinitival clause embedding, we can have any number of verbal heads with the arguments "jumping up" to the embedding clauses from an arbitrarily deeply embedded infinitival clause, as shown in the example below (all the sentences of this example mean '. . . that no-one has tried to promise the customer to repair the refrigerator'):[1]

. . . dass niemand [[dem Kunden] [[den Kühlschrank] zu reparieren] zu versprechen] versucht hat;

. . . dass niemand [den Kühlschrank]$_i$ [[dem Kunden] [t_i zu reparieren] zu versprechen] versucht hat;

. . . dass [den Kühlschrank]$_i$ niemand [[dem Kunden] [t_i zu reparieren] zu versprechen] versucht hat;

. . . dass [dem Kunden]$_j$ niemand [t_j [[den Kühlschrank] zu reparieren] zu versprechen] versucht hat;

. . . dass [den Kühlschrank]$_i$ [dem Kunden]$_j$ niemand [t_j [t_i zu reparieren] zu versprechen] versucht hat;

. . . dass [dem Kunden]$_j$ [den Kühlschrank]$_i$ niemand [t_j [t_i zu reparieren] zu versprechen] versucht hat.

The string language of scrambled sentences can be seen as $\{n^i v^i \mid n, v \in \Sigma, i > 0\}$, it is context-free. But what matters from the linguistic point of view is not so much the generated language as such, but rather the grammar's capacity to assign linguistically correct structural descriptions to the sentences with scrambling. In (Becker, Rambow, and Niv, 1992) it was proved that unbounded scrambling cannot be derived by linear context-free rewriting systems (LCFRS) and—as a consequence—it cannot be derived by set-local multi-component tree-adjoining grammars (slMCTAG) either.

An important aspect of the unbounded scrambling is that there are some syntactic categories, called *barriers*, beyond which no constituents can scramble. For German it is a tensed clause, for example.

Nonlocal vector TAGs with dominance links and integrity constrains (VTAG-Δ) introduced in (Rambow, 1994) are the only known TAG-based formalism which allows a generalized description of scrambling and is polynomially parsable if some restrictions external to the formalism itself are imposed on the derivation. In its lexicalized version these restrictions are satisfied as a consequence of the lexicalization. Other nonlocal versions of TAGs do not have acceptable computational properties. For example, the word recognition problem for nonlocal MCTAGs with such linguistically meaningful restrictions as lexicalization, limiting the numbers of trees in each tree set to two and imposing dominance links on the trees belonging to one set is NP-complete (Champollion, 2007). This shows that nonlocality, which seems to be necessary for the adequate description of un-

[1] The sentences are based on the examples from German in (Rambow, 1994).

bounded scrambling, is generally very dangerous for the computational properties of the formalism.

Another mildly context-sensitive formalism widely studied in the last ten years are Minimalist Grammars (MG) introduced in (Stabler, 1997) as a formalization of some central aspects of the structure-building component of the Minimalist Program, an approach to the description of syntax proposed in (Chomsky, 1995). In this formalism, discontinuous constituents are described as a result of the displacement of a part of a constituent into some other position in the tree. MGs are weakly equivalent to set-local MCTAGs. In MGs the locality is represented as the *shortest-move constraint* (SMC) forbidding competitive displacement of constituents. Lifting this constraint affects badly the computational properties of the formalism: for example, canceling the SMC, but preserving the *specifier island constraint* (SPIC) prohibiting movement from within specifiers, produces a Turing-equivalent formalism (Kobele and Michaelis, 2005). In (Frey and Gärtner, 2002), a scrambling operator was introduced for MG, but it was restricted by the SMC which made the generalized scrambling description impossible.

In the present paper we show that extending an MG with an unbounded scrambling (i.e., scrambling without SMC) and with barriers—an analogue to the integrity constraints in VTAG-Δ—makes the recognition problem for the resulting formalism NP-hard.

3 MGs with unbounded scrambling and barriers

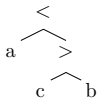
Below we will give a definition of unrestricted Minimalist Grammars with unbounded scrambling and barriers which is based on the original definition of MG in (Stabler, 1997) and (Michaelis, 2001).

Definition 1 ($\mathbf{MG}_B^{scr}$) *An unrestricted Minimalist Grammar with unbounded scrambling and barriers, MG_B^{scr}, is a tuple $G = \langle NonSyn, Syn, c, |, Lex, \Omega \rangle$, such that*

- *$NonSyn$ is a finite set of non-syntactic features partitioned into a set of phonetic ($Phon$) and semantic (Sem) features.*
- *Syn is a finite set of syntactic featured disjount from $NonSynt$ and partitioned into*
  - *a set $B = \{ n, v, d, c, t, \ldots \}$ of base (syntactic) categories,*
  - *a set of abstract features, $A = \{ case, num, pers, \ldots \}$,*
  - *a set of merge selectors, $M = \{ =x \mid x \in B \}$,*
  - *a set of move licensees, $E = \{ -f \mid f \in A \}$,*
  - *a set of move licensors, $R = \{ +f \mid f \in A \}$,*
  - *a set of scramble licensees, $S = \{ \sim x \mid x \in B \}$,*
  - *a set of barrier markers, $I = \{ x \mid x \in B \}$.*
- *c is a distinguished element of B, the completeness category.*
- *'$|$' is a special symbol (a bar).*
- *Lex is a lexicon—a finite set of simple expressions (see Definition 2) over $NonSynt \cup Syn$, each of which is of the form $\tau = \langle N_\tau, \lhd^*, \prec, <, label_\tau \rangle$, with $N_\tau = \{\epsilon\}$.*
- *Ω is the set of the structure-building operations 'merge', 'move' and 'scramble'.*

In what follows, by $[_< a, b\,]$ we will denote a binary tree consisting of the nodes a and b in this very linear order where the node a is the head of ("projects over") the structure represented by this tree so that the expression associated with the tree is the same as the one associated with its head node. In the same way, by $[_> c, b\,]$ we will denote a binary tree consisting of the nodes c and b in this very linear order where the node b is the head of the structure represented by the tree:

A node represented by a single letter will be called a *simple* node. All nodes in the above examples are simple ones. If a node represents a subtree, it will be called a *complex* node, as in the following example, where b in the tree $[_< a, b\,]$ is a complex node since it represents its subtree $[_> c, b\,]$:



The argument position to the right of a head node is called the *complement* position. Positions to the left of a head node, over which

this node projects, are referred to as *specifier* positions. The *maximal projection* of a node a in a given tree is the maximal subtree headed by this node.

Definition 2 (Expression) *An expression is a finite, binary, labeled ordered tree* $\tau = (N_\tau, \lhd^*, \prec, <, label_\tau)$, *where*
N_τ *is the set of nodes;*
$\lhd$ *is the dominance relation between nodes;*
$\prec$ *is the precedence relation between nodes;*
$<$ *is the projection relation between nodes;*
$label_\tau$ *is the leaf-labeling function mapping the leafs of the tree onto an element from*
$\{M^*\ R^?\ B\ E^?\ S^?\ Phon^*\ Sem^*\mid\}\cup$
$\{M^*\ R^?\ B^{-I}\ E^?\ S^?\ Phon^*\ Sem^*\mid\}\cup$
$\{E^?\ S^?\ Phon^*\ Sem^*\mid B\}\cup$
$\{E^?\ S^?\ Phon^*\ Sem^*\mid B^{-I}\}$
as introduced in the definition of MG_B^{scr}.
An expression is called complex if it has more than one node; otherwise it is called simple.

An expression τ over $Syn\cup NonSyn$ is called *well-labeled* if each leaf of τ is a string from $Syn^*Phon^*Sem^*(|(B+B^{-I}))^?$. The label of a complex expression is that of its head leaf.

The phonetic yield of an expression is the concatenation of the phonetic yields of its subexpressions.

We will be saying that the expression $e = f_1\, f_2 \ldots f_{n-1} \mid f_n$, where f_1, f_2, …, f_n are features, *has* or *contains* these features and *displays* feature f_1. We will say that a syntactic feature f is *canceled* from the expression e if it is removed from it. We will also say that a syntactic feature f is *hidden* in the expression e if it is moved to the right of the bar symbol in this expression. To make notation shorter, we will omit the bar symbol if there are no features behind it.

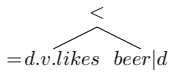
Now we will define the structure-building operations with their domains.

Definition 3 (*merge* domain)
$Dom(merge) = \{\ \langle\tau_0, \tau\rangle \mid \tau_0$ *and* τ *are well-labeled expressions,* τ_0 *displays category* x, *and* τ *displays feature* $=x\ \}$.

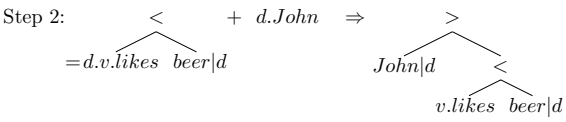
Definition 4 (*merge* operator)
$merge(\tau) = [_<\tau', \tau_0']$, *such that*
τ *is a simple node displaying feature* $=x$,
τ_0 *displays category* x,
τ' *is like* τ *except that* $=x$ *is canceled,*
τ_0' *is like* τ_0 *except that* x *is hidden;*
and $merge(\tau) = [_>\tau_0', \tau']$, *such that*
τ *is a complex node that displays feature* $=x$,
τ_0 *displays category* x,
τ' *is like* τ *except that* $=x$ *is canceled,*
τ_0' *is like* τ_0 *except that* x *is hidden.*

As an example of *merge* we will consider the derivation of the sentence *John likes beer.*
Lexicon: $=d.=d.v.likes$; $d.John$; $d.beer$
Derivation:

Step 1: $=d.=d.v.likes\ +\ d.beer \Rightarrow$



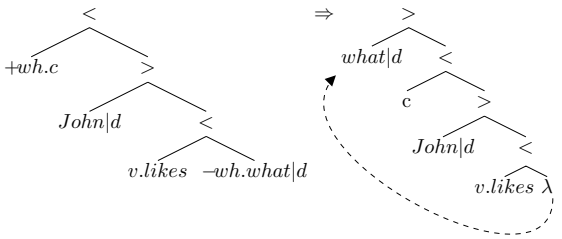
Step 2:



Definition 5 (*move* domain)
$Dom(move) = \{\ \tau \mid \tau$ *is a well-labeled expression that displays feature* $+x$ *and contains exactly one maximal projection* τ_0 *displaying feature* $-x\ \}$.[2]

Definition 6 (*move* operator)
$move(\tau) = [_>\tau_0', \tau']$, *such that*
τ *displays feature* $+x$,
τ_0 *is a proper subtree of* τ *displaying feature* $-x$,
τ_0' *is like* τ_0 *except that* $-x$ *is canceled, and*
τ' *is like* τ *except that* $+x$ *is canceled and the subtree* τ_0 *is replaced by an empty leaf.*

The operator *move* is illustrated below in the derivation of the subordinate clause *what John likes* from *John likes what* within the sentence *she wonders what John likes.*
Lexicon: $=d.=d.v.likes$; $d.John$; $d.-wh.what$; $=v.+wh.c$
Derivation:



We say that a maximal projection τ' is a *barrier* between the maximal projections τ and τ_0, if τ_0 is a proper subtree of τ', τ' is a proper subtree of τ, τ_0 has the basic category b, and τ' contains the barrier marker $-b$.

[2] The restriction that τ cannot contain more than one movement candidate is the shortest-move condition, as it is used in MG.

Definition 7 (*scrambling* domain)
$Dom(scramble) = \{ \tau \mid \tau$ is a well-labeled expression that displays category x and contains at least one maximal projection τ_0 displaying feature $\sim x$ and there is no barrier between τ and $\tau_0 \}$.

Definition 8 (*scrambling* operator)
$scramble(\tau) = [_{>}\tau'_0, \tau']$, such that
τ displays category x,
τ_0 is a proper subtree of τ displaying feature $\sim x$ and there is no barrier between τ and τ_0,
τ'_0 is like τ_0 except that $\sim x$ is canceled,
τ' is like τ except that subtree τ_0 is replaced with an empty leaf.

The scrambling so defined operates nondeterministically in the sense that it can displace any appropriate constituent. The difference between scrambling and movement consists in the fact that scrambling is optional, it allows a competitive displacement of constituents since it is not restricted by SMC, and it can be blocked by a barrier.

Definition 9 (Language of an MG_B^{scr})
The language L generated by an MG_B^{scr} G is the set of the phonetic yields of the expressions produced from the lexical entries by applying (some of) the structure-building operations, such that these expressions display the completeness category c and neither they themselves nor their subexpressions contain move licensees and move licensors (i.e., all movements have been performed).

4 MG_B^{scr} is NP-hard

4.1 Some preliminaries

A problem X is NP-hard if and only if an NP-complete problem N can be transformed ("reduced") to X in polynomial time in such a way that a (hypothetical) polynomial-time algorithm solving X could also be used to solve N in polynomial time.

For a language L, we will denote by $L()$ the word recognition problem for L. Let L, L_1 and L_2 be languages such that $L = L_1 \cup L_2$ and $L_1 \cap L_2 = \emptyset$. Let $p(w)$ be a polynomial-time computable function such that for any $w \in L$ it returns *true* if $w \in L_1$ and $false$ otherwise. (For a $w \notin L$, it can return either *true* or $false$.) We will need following proposition:

Proposition 1 *If $L_1()$ is NP-hard, then $L()$ is also NP-hard.*

4.2 The idea of the proof

The NP-hardness of the word recognition problem for MG_B^{scr} will be proved by constructing a grammar $G \in MG_B^{scr}$ that generates a language $L = L_1 \cup L_2$, $L_1 \cap L_2 = \emptyset$, where $L_1()$ represents a known NP-complete problem, i.e., it is NP-hard, and the question whether a word $w \in L$ belongs to L_1 or to L_2 can be resolved in deterministic polynomial time. In the proof we will use the 3-Partition Problem which in known to be (strongly) NP-complete:

> Given a set of $3k$ natural numbers $\{n_1, n_2, \ldots, n_{3k}\}$ and a constant m, decide whether this set can be partitioned into k subsets of cardinality 3 each of which sums up to m.

This problem can be described as a language

$$L_{3P} = \{b^m a x^{n_1} a x^{n_2} \ldots a x^{n_{3k}} \mid a, b, x \in \Sigma\}$$

such that it consists of *all* the words for which $\langle m, n_1, n_2, \ldots, n_{3k}\rangle$ represents an instance of the problem. The word recognition problem for this language is NP-hard.[3]

In MG_B^{scr}, scrambling allows syntactic constituents to move to the left in competitive manner while barriers set boundaries beyond which these constituents cannot move. This fact can be used to derive a language L_B^{scr} containing L_{3P} such that for any word $w \in L_B^{scr}$ it can be decided in deterministic polynomial time whether $w \in L_{3P}$ or not.

4.3 Proving NP-hardness

Let $G = \langle NonSyn, Syn, p, |, Lex, \Omega\rangle$ be an MG_B^{scr} where

- $Phon = \{a, b, c, d\}$, $Sem = \emptyset$,
- $A = \{ f \}$, and
- $B = \{ a_1, a_2, a_3, a'_1, a'_2, a'_3, a''_1, a''_2, a''_3, b, b', b^0, c_1, c_2, c_3, c'_1, c'_2, c'_3, c''_1, c''_2, c''_3, d_1, d_2, d_3, d'_1, d'_2, d'_3, d''_1, d''_2, d''_3, e, g, s, p \}$.

The lexicon of the grammar, Lex, consists of the following entries (organized into groups

[3] A language representation of the 3-Partition Problem was also used in (Champollion, 2007) to prove NP-hardness for a restricted version of nonlocal MCTAGs. It should be mentioned, though, that the relationship between nonlocal MCTAGs and MG_B^{scr} is not known, so we cannot apply the complexity result for nonlocal MCTAGs to MG_B^{scr}.

according to which part of the structure they generate):

1. (a) $=c''_3 \,.\, a''_3 \,.\, \sim s \,.\, a$; $=d''_3 \,.\, = b^0 . \, c''_3 \,.\, c$;
   $=c''_3 \,.\, = b \,.\, d''_3 . \, d$; $=e \,.\, = b . \, d''_3 \,.\, d$; e;
   (b) $=c''_2 \,.\, a''_2 \,.\, \sim s \,.\, a$; $=d''_2 \,.\, = b^0 . \, c''_2 \,.\, c$;
   $=c''_2 \,.\, = b \,.\, d''_2 \,.\, d$; $=a''_3 \,.\, = b \,.\, d''_2 \,.\, d$;
   (c) $=c''_1 \,.\, a''_1 \,.\, \sim s \,.\, a$; $=d''_1 \,.\, = b^0 . \, c''_1 \,.\, c$;
   $=c''_1 \,.\, = b \,.\, d''_1 \,.\, d$; $=a''_2 \,.\, = b \,.\, d''_1 \,.\, d$;
2. (a) $=c_3 \,.\, a_3 \,.\, \sim s \,.\, a$; $=d_3 \,.\, c_3 \,.\, c$;
   $=c_3 \,.\, = b' . \, d_3 \,.\, d$; $=a'_1 \,.\, = b' . \, d_3 \,.\, d$;
   $=a''_1 . \, = b' . \, d_3 \,.\, d$;
   (b) $=c_2 \,.\, a_2 \,.\, \sim s \,.\, a$; $=d_2 \,.\, c_2 \,.\, c$;
   $=c_2 \,.\, = b' . \, d_2 \,.\, d$; $=a_3 \,.\, = b' . \, d_2 \,.\, d$;
   (c) $=c_1 \,.\, a_1^{-b} . \, \sim s \,.\, a$; $=d_1 \,.\, c_1 \,.\, c$;
   $=c_1 \,.\, = b' . \, d_1 \,.\, d$; $=a_2 \,.\, = b' . \, d_1 \,.\, d$;
3. (a) $=c'_3 \,.\, a'_3 \,.\, \sim s \,.\, a$; $=d'_3 \,.\, c'_3 \,.\, c$;
   $=c'_3 \,.\, = b \,.\, d'_3 \,.\, d$; $=a_1 \,.\, = b \,.\, d'_3 \,.\, d$;
   (b) $=c'_2 \,.\, a'_2 \,.\, \sim s \,.\, a$; $=d'_2 \,.\, c'_2 \,.\, c$;
   $=c'_2 \,.\, = b \,.\, d'_2 \,.\, d$; $=a'_3 \,.\, = b \,.\, d'_2 \,.\, d$;
   (c) $=c'_1 \,.\, a'^{-b'}_1 . \, \sim s \,.\, a$; $=d'_1 \,.\, c'_1 \,.\, c$;
   $=c'_1 \,.\, = b \,.\, d'_1 \,.\, d$; $=a'_2 \,.\, = b \,.\, d'_1 \,.\, d$;
4. $=a_1 \,.\, g \,.\, -f$; $=a'_1 . \, g \,.\, -f$; $=a''_1 . \, g \,.\, -f$;
5. $=g \,.\, s$;
6. $=s \,.\, +f \,.\, p$;
7. $b \,.\, \sim c \,.\, b$; $b' . \, \sim c' . \, b$; $b \,.\, \sim g \,.\, b$; $b' . \, \sim g \,.\, b$; $b^0 . \, b$

Proposition 2 *The language L generated by the grammar G is a union of two disjoint languages, $L = L_{3p} \cup L'$, $L_{3p} \cap L' = \emptyset$, such that L_{3p} consists of all the words*

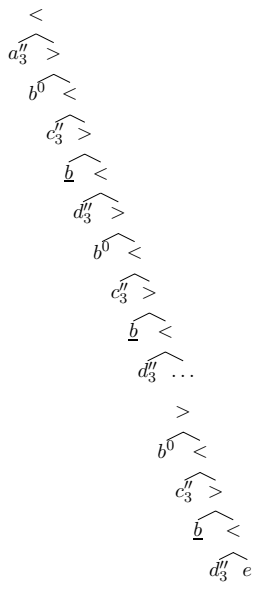
$$b^m a(bcd)^{n_1} a(bcd)^{n_2} \ldots a(bcd)^{n_{3k}}$$

with $a, b, c, d \in \Sigma$, where $\langle m, n_1, n_2, \ldots, n_{3k} \rangle$ is an instance of the 3-Partition Problem, as described above, and there exists a polynomial-time computable function $p(w)$ such that for any word $w \in L$ it returns true if $w \in L_{3p}$ and $false$ otherwise; for $w \notin L$ it returns either true of $false$.

We will prove the proposition 2 by following the bottom-up derivation of the language L. In the illustrations below, the symbols used in the tree structures are base category symbols.[4] The derivation starts at step 1.

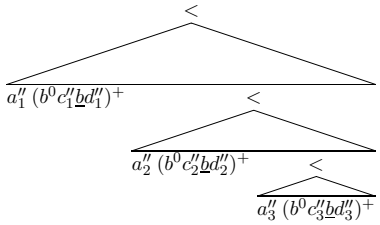
[4] In the grammar G, the lexical entries are made in such a way that the phonetic (i.e., terminal) symbols can be obtained by stripping the base category symbols of indices and bars (except for the zero-yield entries headed by e, g, s and p).

Step 1. The derivation begins with the lexical entries (1a) generating the following (sub)tree:



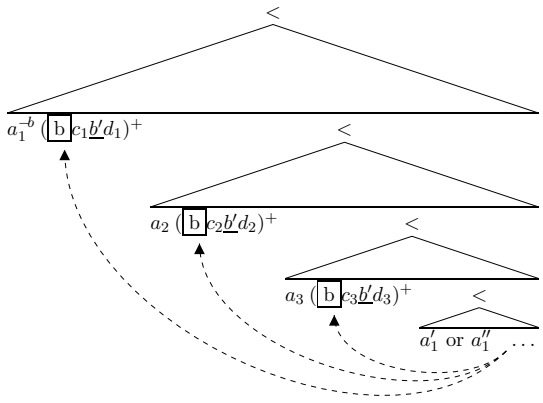
The yield of this subtree is $a(bcbd)^+$. Each b located immediately between a c and a d (the corresponding base category is underlined) is licensed for scrambling to a specifier position of a c or g introduced at a later point in the derivation, since every such b has the scrambling licensee $\sim c$ or $\sim g$. The whole a''_3-headed subtree is licensed for scrambling to the s node to be introduced at a later point in the derivation, since the a''_3 node has the scrambling licensee $\sim s$.

After that, subtrees headed with a''_2 and a''_1 are generated by the entries (1b) and (1c) respectively. The generation proceeds in the same way as in the case of the a''_3 subtree; the b nodes are licensed for scrambling to c or g, and the a''_2 and a''_1 subtrees are themselves licensed for scrambling to s:



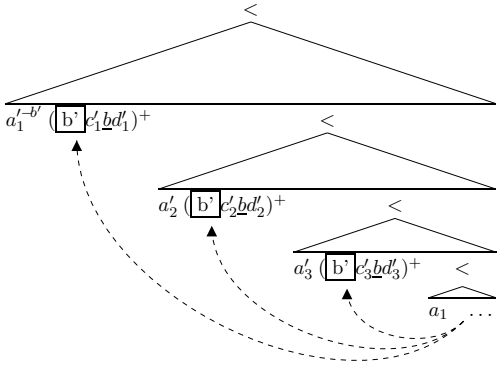
The phonetic yield generated at this point is $a(bcbd)^+ a(bcbd)^+ (bcbd)^+$. The derivation continues to step 2 or 4.

Step 2. Analogously to the previously performed step, subtrees headed by a_3, a_2 and a_1 are generated by the entries (2a), (2b) and (2c) respectively. All of them are licensed for scrambling to s. The b' nodes inside these subtrees are licensed for scrambling to c' or g. Some of the b nodes *introduced in the previously performed step* (this restriction is provided by barriers) scramble to some of the c nodes introduced at the present step:



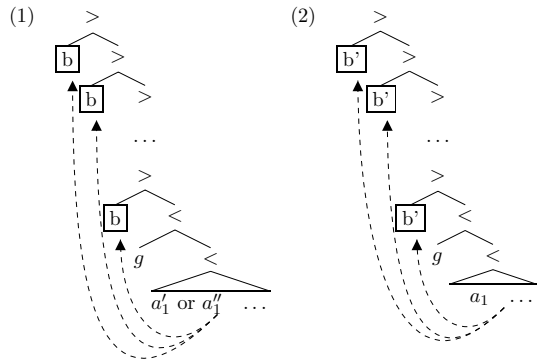
The derivation continues to step 3 or 4.

Step 3. Analogously to the previously performed step, subtrees headed by a_3', a_2' and a_1' are generated by the entries (3a), (3b) and (3c) respectively. All of them are licensed for scrambling to s. The b nodes inside these subtrees are licensed for scrambling to c or g. Some of the b' nodes *introduced in the previously performed step* (this restriction is provided by barriers) scramble to some of the c' nodes introduced at the present step:



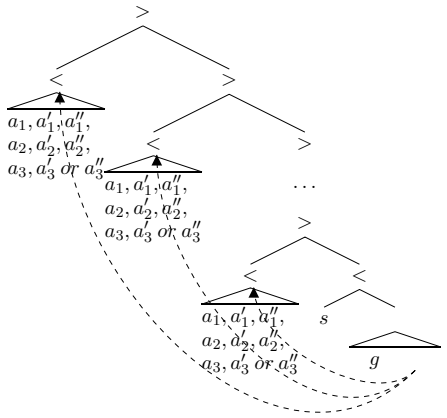
The derivation continues to step 2 or 4.

Step 4. A subtree headed by g is generated by the entries (4). The g head takes as its complement a_1' or a_1'' (1), or a_1 (2). It is licensed for movement to p. Some of the b' or b nodes *introduced in the previously performed step* (this restriction is provided by barriers) scramble to g:



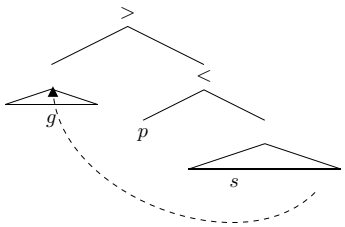
The derivation continues to step 5.

Step 5. A subtree headed by s is generated by the entry (5). The s head takes g as its complement. Further, some a subtrees generated at previous steps scramble to s:



The derivation continues to step 6.

Step 6. A subtree headed by p is generated by the entry (6). The p head takes s as its complement. Further, the g subtree generated at a previous step is moved to the specifier position of p:



The language generated by this grammar, L, is the union of two languages, $L = L_1' \cup L_1$, such that L_1' consists of all the words produced with all b and b' nodes having scrambled and each c and c' head having accepted *exactly one* scrambling b or b' node, and L_1 contains the rest of the words. The language L_1' consists of all the words

$$b^m\, a(bcd)^{n_1}\, a(bcd)^{n_2} \ldots a(bcd)^{n_{3k}}$$

such that for all positive natural numbers k and m, the multiset $\{n_1, n_2, \ldots, n_{3k}\}$ can be partitioned into k multisets of cardinality 3, each of which sums to m. This will be explained following the generation of the words of the language. On the yield level, each "a-tripple" $a(\,\mathrm{b}\,c\underline{b}d)^+ a(\,\mathrm{b}\,c\underline{b}d)^+ a(\,\mathrm{b}\,c\underline{b}d)^+$ generated at the step (2) or (3) receives the scrambling symbols b from the neighbouring

a-tripple on the right (these symbols are depicted in squares) generated during the previous step and later "gives away" through scrambling to the neighbouring left a-tripple the symbols b located between c and d (underlined). Barriers guarantee that these symbols can only scramble to the adjacent tripple. The symbols b scrambling from the leftmost a-tripple are stored as a "counter" at step (4). In case all b and b' symbols have scrambled and each c and c' head have received through scrambling exactly one b or b', all a-tripples will contain an equal number of bcd subwords, while the number of these subwords in each $a(bcd)^+$ member of one and the same a-tripple may vary. The "counter" will consist of as many symbols b as there are bcd subwords in each a-tripple. At step (5), all the $a(bcd)^+$ members of the a-tripples are permuted arbitrarily, whereafter the "counter subword" is moved to the left at step (6).

Each word in L_1 contains at least one following subword in positions to the right starting from the leftmost occurrence of a: bb (more than one b have scrambled to the same c head), ac, dc (omission of scrambling to a particular c head), cb (b has not scrambled), while no word in L'_1 follows this pattern. This means that $L'_1 \cap L_1 = \emptyset$, and there exists a *polynomial-time computable* function $p(w)$ such that for any $w \in L$, $p(w) = true$ if $w \in L'_1$ and $p(w) = false$ otherwise. For a $w \notin L$, it will return $true$ or $false$.

The language L'_1 can be seen as a union of two languages, $L'_1 = L_2 \cup L_3$, such that $\{n_1, n_2, \ldots, n_{3k}\}$ is a *proper multiset* for L_2 (i.e., it contains repeated elements) and a *set* for L_3. This means that $L_2 \cap L_3 = \emptyset$, and—since the problem whether a given multiset is a proper multiset or a set can be solved in deterministic polynomial time—there exists a *polynomial-time computable* function $q(w)$ such that for any $w \in L'_1$, $q(w) = true$ if $w \in L_3$ and $q(w) = false$ otherwise. For a $w \notin L'_1$, it will return $true$ or $false$.

The language L_3 constitutes the unary encoding of the 3-Partition Problem[5] whereby we have proved the proposition 2, which together with the proposition 1 gives us following result:

Proposition 3 *The word recognition problem for MG_B^{scr} is NP-hard.*

[5]Without loss of generality we consider only positive natural numbers and assume $k \geq 1$.

5 Conclusions

Since the recognition problem for MG_B^{scr} is NP-hard, the generalized description of scrambling is probably impossible in MG, at least if it is implemented in a straightforward way. On the other hand, MGs can provide a convenient framework for the practical implementation of some important results obtainable within the Minimalist Program. For this reason, a further study of the proposed MG extensions is important, since a solution to the scrambling problem can make out of MGs a powerful formal language tool for the grammar engineering. Additionally, it could provide insights into possible ways to tackle the nonlocality problem in this class of formalisms.

References

Becker, T., O. Rambow, and M. Niv. 1992. The Derivational Generative Power of Formal Systems or Scrambling is Beyond LCFRS. Technical Report IRCS-92-38, University of Pennsylvania, USA.

Champollion, L. 2007. Lexicalized nonlocal MCTAG with dominance links is NP-complete. In *Proceedings of Mathematics of Language 10. To appear.*

Chomsky, N. 1995. *The Minimalist Program.* The MIT Press, Cambridge, USA.

Frey, W. and H.-M. Gärtner. 2002. On the Treatment of Scrambling and Adjunction in Minimalist Grammars. In G. Jäger, P. Monachesi, G. Penn, and S. Wintner, editors, *Proceedings of Formal Grammar 2002*, pages 41–52, Trento, Italy.

Kobele, G. M. and J. Michaelis. 2005. Two Type 0-Variants of Minimalist Grammars. In *Proceedings of the 10th conference on Formal Grammar and the 9th Meeting on Mathematics of Language*, Edinburgh, Scotland.

Michaelis, J. 2001. *On Formal Properties of Minimalist Grammars.* Ph.D. thesis, Potsdam University, Germany.

Rambow, O. 1994. *Formal and Computational Aspects of Natural Language Syntax.* Ph.D. thesis, University of Pennsylvania, USA.

Stabler, E. 1997. Derivational minimalism. In Christian Retore, editor, *Logical Aspects of Computational Linguistics.* Springer, pages 68–95.