# Scrambling as the Combination of Relaxed Context-Free Grammars in a Model-Theoretic Grammar Formalism

Ralph Debusmann
Programming Systems Lab
Universität des Saarlandes
Postfach 15 11 50
66041 Saarbrücken, Germany
rade@ps.uni-sb.de

## 1 Introduction

Five years after the first ESSLLI workshop on Model-Theoretic Syntax (MTS), Pullum and Scholz (2001) stated that since the work on MTS had largely focused on reformulating existing GES frameworks, in a sense, it had been done in the shadow of Generative-Enumerative Syntax (GES).

In the following five years, the bulk of work has still been invested in model-theoretic reformulations of GES frameworks. Reformulations of GB can be found in (Rogers, 1996, 2003), of LFG in (Blackburn and Gardent, 1995), of GPSG in (Kracht, 1995) and (Rogers, 1996, 2003), of HPSG in (Kepser, 2000) and (Kepser and Mönnich, 2003), and of TAG in (Rogers, 2003).

Recently (Rogers, 2004), there have been attempts to step out of the shadow of GES, and to use MTS not only to reformulate and compare existing frameworks, but to utilize the more declarative, clarifying perspective of MTS to also explore *extensions* of them. This is what we set out to do in this paper as well.

We base our work on the model-theoretic meta grammar formalism of Extensible Dependency Grammar (XDG) (Debusmann, 2006). XDG can be used to axiomatize grammatical theories based on dependency grammar, to extend them, and to implement them using the constraint-based XDG Development Kit (XDK) (Debusmann et al., 2004), (Debusmann and Duchier, 2007). XDG is novel in supporting the axiomatization of *multi-dimensional* grammatical theories, where the linguistic aspects of e.g. syntax and semantics can be modeled modularly by separate dependency analyses.

This paper contributes a new, previously unpublished formalization of XDG in first-order logic (section 2), and the first results on the closure properties of the string languages licensed by XDG (section 3). The closure properties are proven based on the operation of *grammar composition*, where the string language resulting from the composition of two grammars $G_1$ and $G_2$ is the difference, union or intersection of that of $G_1$ and $G_2$.

In section 4, we recap the axiomatization of Context-Free Grammar (CFG) of (Debusmann,

2006), which we employ as our launch pad to go beyond CFG in section 5. First, we explore the *relaxation* of the contiguity criterion of CFG, and second, we explore the *intersection* of CFGs. This brings us into the position to formulate a simple and elegant account of German scrambling loosely based on (Duchier and Debusmann, 2001).

## 2 Extensible Dependency Grammar

XDG models tuples of dependency graphs sharing the same set of nodes, which are anchored by the same string of words. The components of the tuple are called *dimensions*, and XDG analyses *multigraphs*.

Figure 1 shows an example multigraph with two dimensions: SYN provides a syntactic, and SEM a semantic analysis in terms of predicate-argument structure. The nodes are identified by indices (1 to 6), and associated with words (e.g. *Mary*, *wants*, etc.). The edge labels on SYN are subj for "subject", vinf for "full infinitive", part for "particle", obj for "object" and adv for "adverb". On SEM, ag stands for "agent", pat for "patient" and th for "theme".

Contrary to other dependency-based grammar formalisms such as (Gaifman, 1965), XDG dimensions need not be projective trees, but can in fact be general graphs as in Word Grammar (Hudson, 1990). An example is the SEM dimension in Figure 1, which is not a tree but a directed acyclic graph (DAG). Here, *to*, which does not have any semantic content, has no ancestor, and *Mary*, which is the agent of both *wants* and *eat*, has two.

Multigraphs are constrained by *grammars* specifying:

1. A *multigraph type* determining the possible dimensions, words, edge labels and additional attributes associated with the nodes called *node attributes*.

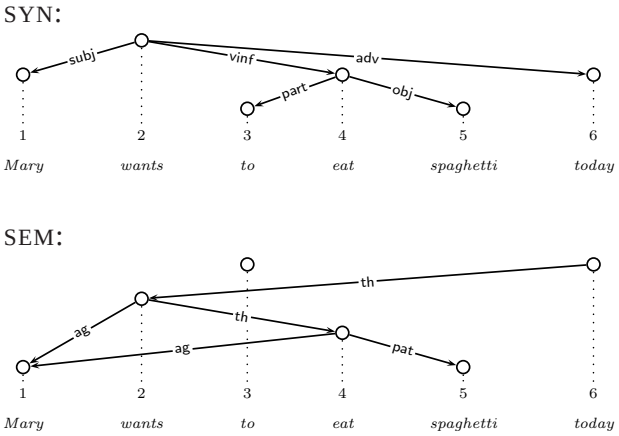



Figure 1: XDG multigraph for *Mary wants to eat spaghetti today*.

2. A *lexicon* determining a subset of the node attributes of each node, depending on the associated word.

3. A set of *principles* stipulating the well-formedness conditions of the multigraphs.

XDG is a *meta* grammar formalism. *Instances* of XDG are defined by fixing a multigraph type and a set of principles, and leaving the lexicon variable.

XDG principles stipulate e.g. treeness, DAG-ness, projectivity, valency and order constraints. They can also constrain the relation of multiple dimensions, which is used e.g. in the linking principle to constrain the relation between arguments on SEM and their syntactic realization on SYN. Some principles are *lexicalized*, i.e., they constrain the analysis with respect to the lexicon.

The lexicon constrains all dimensions simultaneously, and thereby synchronizes them. Figure 2 depicts an example graphical lexical entry for the word *eat*. On SYN, by the lexicalized valency principle, the lexical entry licenses zero or one incoming edges labeled vinf, precisely one part, zero or one obj, ar-

bitrary many adv dependents, and no other incoming and outgoing edges. By the order principle, the part dependents must precede the head *eat*, which must precede the obj and the adv dependents. On SEM, the lexical entry licenses arbitrary many incoming th edges, and requires precisely one ag dependent and zero or one pat dependents (valency principle). It licenses no other incoming and outgoing edges. The patient must be realized by the object (linking principle). The realization of the agent is not constrained.

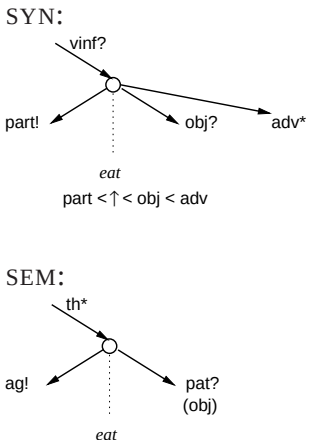



Figure 2: Lexical entry for the word *eat*

## 2.1 Multigraph

We turn to the formalization of XDG. Contrary to (Debusmann, 2006), which is higher-order, our formalization is first-order, and hence called FO XDG. We begin with multigraphs. Multigraphs are formulated over the *labeled dominance relation*. This corresponds to the transitive closure of the labeled edge relation, where the label is the label of the first edge. The purpose of including this relation and not the labeled edge relation itself is to stay in first-order logic: if we included only the labeled edge relation, we could not express the transitive closure without extending the logic with fixpoints or second-order quantification.

**Definition 1** (Multigraph)**.** *Given a finite set of dimensions $D$, a finite set of words $W$, a finite set of edge labels $L$, a finite set of attributes $A$, and a finite set of set types $T$, a multigraph $M = (V, E^+, <, nw, na)$ consists of a finite set of nodes $V$, the set of labeled dominances $E^+ \subseteq V \times V \times L \times D$, a total order $< \subseteq V \times V$ on the set of nodes, the node-word mapping $nw \in V \to W$, and the node-attributes mapping $na \in V \to D \to A \to \cup\{u \mid u \in T\}$. We define $V$ as a finite interval of the natural numbers starting with 1. $(v, v', l, d) \in E^+$ iff on dimension $d$, the multigraph contains an edge from $v$ to $v''$ labeled $l$, and a path of arbitrary many edges from $v''$ to $v'$ with any labels.*

## 2.2 Grammar

**Definition 2** (Grammar)**.** *A grammar $G = (MT, lex, P)$ consists of a multigraph type $MT$, a lexicon $lex$, and a set of principles $P$.*

**Definition 3** (Multigraph Type)**.** *Given a set of atoms $At$, a multigraph type $MT = (D, W, L, dl, A, T, dat)$ consists of a finite set of dimensions $D \subseteq At$, a finite set of words $W \subseteq At$, a finite set of labels $L \subseteq At$, a dimension-label mapping $dl \in D \to 2^L$, a finite set of attributes $A \subseteq At$, a finite set of types $T \subseteq Ty$, and a dimension-attributes-type mapping $dat \in D \to A \to T$. $Ty$ is the set of types built from finite domains $Fd$: $Ty ::= 2^{Fd_1 \times \ldots \times Fd_n}$, where $Fd ::= V \mid \{a_1, \ldots, a_n\}$, $V$ is a placeholder for the set of nodes, and $a_1, \ldots, a_n \in At$.*

**Definition 4** (Multigraph of Multigraph Type)**.** *A multigraph $M = (V, E^+, <, nw, na)$ is of multigraph type $MT = (D, W, L, dl, A, T, dat)$ iff the sets of dimensions $D$, words $W$, edge labels $L$, attributes $A$ and types $T$ match, all labeled dominances on dimension $d \in D$ have only edge labels in $dl\ d$, and*

*all node attributes $a \in A$ on dimension $d \in D$ have a value in dat d a.*

**Definition 5** (Lexicon)**.** *The lexicon is a function from words to sets of lexical entries: $lex \in W \rightarrow 2^{D \rightarrow A' \rightarrow \cup\{u \mid u \in T'\}}$, where $A' \subseteq A$ is the subset of lexical attributes, and for all $w \in W$, if $e \in lex\ w$, then for all $d \in D$, $a \in A'$, $(e\ d\ a)$ has a value in $(dat\ d\ a)$. $T' \subseteq Ty'$, where $Ty'$ is the set of types built from finite domains $Fd'$: $Ty' ::= 2^{Fd'_1 \times \ldots \times Fd'_n}$, where $Fd' ::= \{a_1, \ldots, a_n\}$.*

That is, whereas non-lexical attributes *can* talk about nodes in the multigraph, lexical attributes cannot, since the set of nodes is unknown at the time of lexicon creation.

**Definition 6** (Principles)**.** *Principles are a finite set $P \subseteq \phi$ of first-order formulas built from terms $t ::= c \mid x$, where c is an individual constant and x an individual variable. $\phi$ is defined as follows:*

$$\phi \quad ::= \quad \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x : \phi \mid t_1 = t_2 \mid \psi$$

*where the* predicates $\psi$ *are defined further below. We define the usual logical operators ($\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists!$, $\neq$) as syntactic sugar, and allow to use variables other than x for convenience (e.g. v for nodes, l for labels, w for words and a for attributes etc.). The constants and predicates of the logic are defined with respect to a multigraph type $MT = (D, W, L, dl, A, T, dat)$. The constants are taken from the set C:*

$$\begin{aligned} C \quad = \quad & D \cup W \cup L \cup A \cup \\ & \{Fd_i \mid 2^{Fd_1 \times \ldots \times Fd_n} \in T,\ 1 \leq i \leq n\} \cup \mathbb{N} \end{aligned}$$

*where $\mathbb{N}$ is the set of natural numbers. The universe of the logic is defined given a multigraph $M = (V, E^+, <, nw, na)$, and equals C with the exception that $\mathbb{N}$ is replaced by V, the actual set of nodes. All constants are interpreted by the identity function. As the universe contains only the nodes of the given multigraph, only this finite subset of the natural numbers can be interpreted, i.e., a principle mentioning node 42 can only be interpreted with respect to a multigraph with at least 42 nodes. Here are the predicates $\psi$:*

$$\begin{aligned} \psi \quad ::= \quad & v \xrightarrow{l}_d \rightarrow^*_d v' \\ \mid \quad & v < v' \\ \mid \quad & (W\ v) = w \\ \mid \quad & (t_1 \ldots t_n) \in (d\ v).a \end{aligned}$$

*where $v \xrightarrow{l}_d \rightarrow^*_d v'$ is interpreted as the labeled dominance relation, i.e., $(v, v', l, d) \in E^+$ and $v < v'$ by the total order $<$, i.e., $(v, v') \in <$. $(W\ v) = w$ is interpreted by the node-word mapping, i.e., $nw\ v = w$, and $(t_1 \ldots t_n) \in (d\ v).a$ by the node-attributes mapping, i.e., $(t_1, \ldots, t_n) \in na\ v\ d\ a$.*

*For convenience, we define shortcuts for strict dominance (with any label), labeled edge and edge (with any label):*

$$\begin{aligned} v \rightarrow^+_d v' \quad & \stackrel{\text{def}}{=} \quad \exists l : v \xrightarrow{l}_d \rightarrow^*_d v' \\ v \xrightarrow{l}_d v' \quad & \stackrel{\text{def}}{=} \quad v \xrightarrow{l}_d \rightarrow^*_d v' \wedge \neg\exists v'' : v \rightarrow^+_d v'' \wedge v'' \rightarrow^+_d v' \\ v \rightarrow_d v' \quad & \stackrel{\text{def}}{=} \quad \exists l : v \xrightarrow{l}_d v' \end{aligned}$$

*where we define labeled edge as labeled dominance between v and $v'$ with the restriction that there must be no node $v''$ in between.*

## 2.3 Models

**Definition 7** (Models)**.** *The models of a grammar $G = (MT, lex, P)$, m G, are all multigraphs of multigraph type MT which satisfy the lexicon lex and the principles P.*

**Definition 8** (Lexicon Satisfaction)**.** *Given a grammar $G = (MT, lex, P)$, a multigraph $M = (V, E^+, <, nw, na)$ satisfies the lexicon lex iff for all nodes $v \in V$, there is a lexical entry e for the word*

*of v, and for all dimensions $d \in D$ and all lexical attributes $a \in A'$, the value of the lexical attribute a on dimension d for node v equals the value of the lexical attribute a on dimension d of e:*

$$\forall v \in V : \exists e \in lex\ (nw\ v) : \forall d \in D : \forall a \in A' : (na\ v\ d\ a) = (e\ d\ a)$$

**Definition 9** (Principles Satisfaction)**.** *Given a grammar $G = (MT, lex, P)$, a multigraph $M = (V, E^+, <, nw, na)$ satisfies the principles P iff the conjunction of all principles in P is true.*

## 2.4 String Language

By concatenating the words of its nodes, each multigraph $M = (V, E^+, <, nw, na)$ defines a string $s\ M$:

$$s\ M \quad = \quad nw\ 1\ \ldots\ nw\ |V|$$

**Definition 10** (String Language)**.** *The string language L G of a grammar G is the set of strings of the models of G:*

$$L\ G \quad = \quad \{s\ M \mid M \in m\ G\}$$

The definition already suggests that for parsing, the set of nodes is determined by the input string *s*: there are always as many nodes as words in the input string. Parsing then consists of adding a finite number of edges between these nodes, i.e., crucially, no nodes are added. This so-called *fixed-size assumption* makes XDG parsing amenable to constraint programming (Schulte, 2002), (Apt, 2003), which we indeed use for the parser implementation in the XDG Development Kit (XDK) (Debusmann et al., 2004), (Debusmann and Duchier, 2007).

## 2.5 Recognition Problems

**Definition 11** (XDG Recognition Problem (RP))**.** *Given a grammar G and a string s, is s in L G?*

We distinguish the following three flavors:

1. universal recognition problem (URP): both *G* and *s* are variable
2. fixed recognition problem (FRP): *G* is fixed and *s* is variable
3. instance recognition problem (IRP): the principles are fixed, and the lexicon and *s* are variable

In (Debusmann, 2007), we prove using results from (Vardi, 1982), that the URP is PSPACE-complete, the FRP and IRP are NP-complete.

## 2.6 Example Principles

We present a number of illustrative example principles. For generality, the principles are parametrized by the dimensions that they constrain.

**Tree principle.** Given a dimension *d*, the tree principle stipulates that 1) there must be no cycles, 2) there is precisely one node without a mother (the root), 3) all nodes have zero or one mothers, and 4) all differently labeled subtrees must be disjoint:

$$\begin{array}{l} tree_d = \\ \forall v : \neg(v \rightarrow_d^+ v) \wedge \\ \exists! v : \neg\exists v' : v' \rightarrow_d v \wedge \\ \forall v : (\neg\exists v' : v' \rightarrow_d v) \vee (\exists! v' : v' \rightarrow_d v) \wedge \\ \forall v : \forall v' : \forall l : \forall l' : v \xrightarrow{l}_d \rightarrow_d^* v' \ \wedge\ v \xrightarrow{l'}_d \rightarrow_d^* v' \ \Rightarrow\ l = l' \end{array}$$

**Projectivity principle.** Given a dimension *d*, the projectivity principle forbids crossing edges by stipulating that all nodes positioned between a head and a dependent must be below the head.

$$\begin{array}{l} projectivity_d = \\ \forall v, v' : \\ (v \rightarrow_d v' \ \wedge\ v < v' \Rightarrow \forall v'' : v < v'' \wedge v'' < v' \Rightarrow v \rightarrow_d^+ v'') \wedge \\ (v \rightarrow_d v' \ \wedge\ v' < v \Rightarrow \forall v'' : v' < v'' \wedge v'' < v \Rightarrow v \rightarrow_d^+ v'') \end{array}$$

For example, this principle is violated on the SEM dimension in Figure 1, where *wants* is positioned between *eat* and *Mary*, but is not below *eat*.

To explain the lexicalized valency, order and linking principles, we show an example concrete lexical entry for *eat* in Figure 3, modeling the graphical lexical entry in Figure 2.

**Valency principle.** Given a dimension $d$, the valency principle constrains the incoming and outgoing edges of each node according to the lexical attributes *in* and *out* of type $2^{(dl\ d)\times\{!,+,?,*\}}$, which models the function $(dl\ d) \to \{!,+,?,*\}$ from edge labels on $d$ to *cardinalities*, where ! stands for "one", $+$ for "more than one", ? for "zero or one", and $*$ for "arbitrary many".

$$
\begin{aligned}
&valency_d = \\
&\forall v : \forall l : \\
&((l,!) \in (d\ v).in \Rightarrow \exists! v' : v' \xrightarrow{l}_d v) \wedge \\
&((l,+) \in (d\ v).in \Rightarrow \exists v' : v' \xrightarrow{l}_d v) \wedge \\
&((l,?) \in (d\ v).in \Rightarrow \neg\exists v' : v' \xrightarrow{l}_d v \vee \exists! v' : v' \xrightarrow{l}_d v) \wedge \\
&(\neg(l,!) \in (d\ v).in \wedge \neg(l,+) \in (d\ v).in \wedge \neg(l,?) \in (d\ v).in \wedge \\
&\neg(l,*) \in (d\ v).in \Rightarrow \neg\exists v' : v' \xrightarrow{l}_d v) \wedge \\
&((l,!) \in (d\ v).out \Rightarrow \exists! v' : v \xrightarrow{l}_d v') \wedge \\
&\dots
\end{aligned}
$$

The remaining part of the principle dealing with the outgoing edges proceeds analogously. Given the concrete lexical entry in Figure 3, the principle constrains node *eat* on SYN such that there can be zero or one incoming edges labeled vinf, there must be precisely one part dependent, zero or one obj dependents, arbitrary many adv dependents, and no other incoming or outgoing edges.

**Order principle.** Given a dimension $d$, the order principle constrains the order of the dependents of each node according to the lexical attribute *order* of type $2^{(dl\ d)\times(dl\ d)}$. The *order* attribute models a partial order on *dl d*, where we require that *dl d* includes the special label $\uparrow$. The only purpose of $\uparrow$ is to denote the head the partial order specified by the *order* attribute, which is why the principle also stipulates that there must not be any edges labeled with $\uparrow$.

$$
\begin{aligned}
&order_d = \\
&\forall v : \forall v' : \neg v \xrightarrow{\uparrow}_d v' \wedge \\
&\forall v : \forall l : \forall l' : (l,l') \in (d\ v).order \Rightarrow \\
&\quad (\forall v' : \quad l = \uparrow \quad \wedge \quad v \xrightarrow{l'}_d v' \quad \Rightarrow \quad v < v') \quad \wedge \\
&\quad (\forall v' : \quad v \xrightarrow{l}_d v' \quad \wedge \quad l' = \uparrow \quad \Rightarrow \quad v' < v) \quad \wedge \\
&(\forall v' : \forall v'' : \quad v \xrightarrow{l}_d v' \quad \wedge \quad v \xrightarrow{l'}_d v'' \quad \Rightarrow \quad v' < v'')
\end{aligned}
$$

For instance, given the concrete lexical entry in Figure 3, the order principle orders all part dependents to the left of the head *eat*, and to the left of the obj and adv dependents of *eat*. The head is ordered to the left of its obj and adv dependents, and the obj precede the adv dependents.

**Linking principle.** Given two dimensions $d_1$ and $d_2$, the linking principle requires for all edges from $v$ to $v'$ labeled $l$ on $d_1$ that if there is a label $l' \in (d_1\ v).link$, then there must be a corresponding edge from $v$ to $v'$ labeled $l'$ on $d_2$. The lexical attribute *link* of type $2^{(dl\ d_1)\times(dl\ d_2)}$ models the function $(dl\ d_1) \to 2^{(dl\ d_2)}$ mapping labels on $d_1$ to sets of labels on $d_2$.

$$
\begin{aligned}
&linking_{d_1,d_2} = \\
&\forall v : \forall v' : \forall l : v \xrightarrow{l}_{d_1} v' \Rightarrow \\
&(\exists l' : (l,l') \in (d_1\ v).link \Rightarrow v \xrightarrow{l'}_{d_2} v')
\end{aligned}
$$

This is only one instance of a family of linking principles. Others are presented e.g. in (Debusmann, 2006). In the concrete lexical entry in Figure 3, $d_1 =$ SEM and $d_2 =$ SYN, and the linking principle stipulates e.g. that the patient of *eat* on SEM must be realized by its object on SYN.

## 2.7 Example Grammars

To illustrate how XDG grammars look like, we present two example grammars. The first, $G_1$, mod-

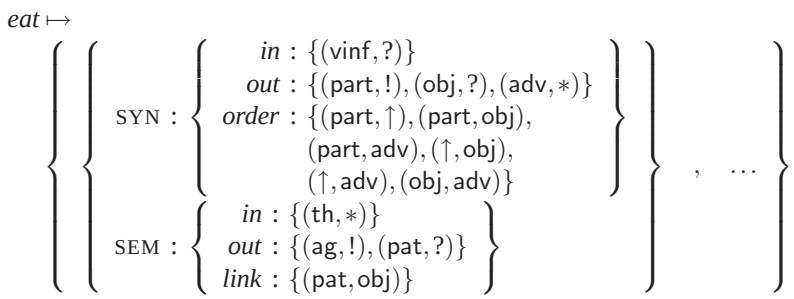



Figure 3: Concrete lexical entry for *eat*

els the string language $L_1$ of equally many *a*s, *b*s and *c*s, in any order:

$$L_1 \quad = \quad \{s \in (a \cup b \cup c)^+ \mid |w|_a = |w|_b = |w|_c\}$$

This grammar demonstrates how to do *counting*. On its sole dimension called ID (for "immediate dominance", in analogy to GPSG), we count using a chain of *a*s, each of which is required to take one *b* and one *c*. An example analysis is depicted in Figure 4. Here, the *a* with index 1 builds a chain with the *a* with index 6. The first *a* takes the *b* with index 3 and the *c* with index 4, and the second *a* the *b* with index 2 and the *c* with index 5.

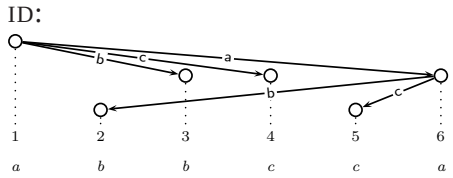



Figure 4: $G_1$ ID example analysis of *a b b c c a*

$G_1$ makes use of the tree principle and the valency principle, where the latter does the counting. The lexicon is depicted graphically in Figure 5. The chain of *a*s is built by the lexical entry for *a* licensing zero or one incoming and outgoing edges labeled a. In addition, we require each *a* to take precisely one b and precisely one c dependent. The lexical entries for *b* and *c* require precisely one incoming edge labeled resp. b and c.

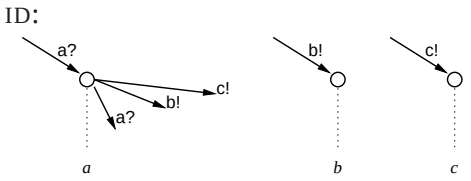



Figure 5: $G_1$ lexical entries for *a*, *b* and *c*

The second example grammar, $G_2$, models the string language $L_2$ of arbitrary many *a*s followed by arbitrary many *b*s followed by arbitrary many *c*s:

$$L_2 \quad = \quad a^+ b^+ c^+$$

With this grammar, we demonstrate how to do *ordering*. On its sole dimension LP (for "linear precedence"), the idea is for the leftmost *a* to be the root, having arbitrary many outgoing edges to arbitrary many other *a*s (labeled 1), and *b*s (2) and *c*s (3) to its right. We show an example analysis in Figure 6.

$G_2$ makes use of the tree, valency and order principles. The lexical entries for the latter two are depicted in Figure 7. Here, the word *a* is lexically ambiguous: it can either be a root (leftmost lexical entry), or a dependent (second from the left). As the

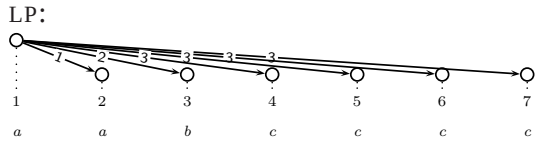



Figure 6: $G_2$ LP example analysis of *a a b b c c*

grammar uses the tree principle, only one *a* will ever become the root, as which it licenses arbitrary many 1 dependents, followed by and one or more 2 dependents, followed by one or more 3 dependents.

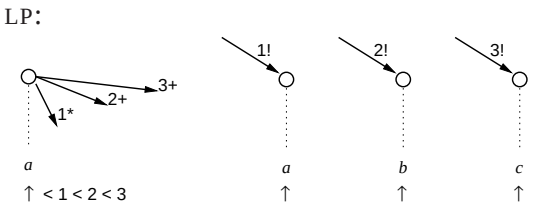



Figure 7: $G_2$ lexical entries for *a*, *b* and *c*

# 3 Closure Properties

In the new formalization presented in section 2, the models and hence also the string languages of XDGs are constrained by FO formulas. This suggests that basic set operations on the string languages of XDGs can all be expressed, and that the XDG string languages are *closed* under these set operations. In this section, we will show that the XDG languages are indeed closed under the following basic set operations:

1. difference and complement
2. union
3. intersection

## 3.1 Grammar Composition

For our proof, we take a detour and define the composition of two XDGs.

**Definition 12** (Grammar Composition)**.** *We define the composition of $G = G_1 \circledcirc G_2$ of $G_1$ and $G_2$ given two grammars $G_1 = (MT_1, lex_1, P_1)$ and $G_2 = (MT_2, lex_2, P_2)$ with multigraph types $MT_1 = (D_1, W_1, L_1, dl_1, A_1, T_1, dat_1)$ and $MT_2 = (D_2, W_2, L_2, dl_2, A_2, T_2, dat_2)$, and a principles composition operator $\circ$.*

*The prerequisites are that 1) the sets of dimensions must be disjoint, and 2) the sets of words must be the same.*

*The resulting grammar $G = G_1 \circledcirc G_2$ with $G = (MT, lex, P)$ has multigraph type $MT = (D, W, L, dl, A, T, dat)$ with:*

$$\begin{aligned} D &= D_1 \cup D_2 \\ W &= W_1 \\ L &= L_1 \cup L_2 \\ dl &= dl_1 \cup dl_2 \\ A &= A_1 \cup A_2 \\ T &= T_1 \cup T_2 \\ dat &= dat_1 \cup dat_2 \end{aligned}$$

*The lexicon lex of G is defined such that lex w contains the product of the lexical entries for w from $G_1$ and $G_2$, for all $w \in W$:*[1]

$$lex\ w \quad = \quad \{e_1 \cup e_2 \mid e_1 \in lex_1\ w \wedge e_2 \in lex_2\ w\}$$

*The principles P of G are defined using the principles composition operator, which combines the conjunction of all principles in $P_1$ and the conjunction of all principles in $P_2$:*

$$P \quad = \quad \{ \bigwedge_{\phi_1 \in P_1} \phi_1 \quad \circ \quad \bigwedge_{\phi_2 \in P_2} \phi_2 \}$$

[1] This clarifies why we demand that $G_1$ and $G_2$ have the same set of words—otherwise, parts of the lexicon of $G$ would be undefined.

Next, we define what it means for a multigraph to be restricted to a subset of the dimensions of its multigraph type.

**Definition 13** (Multigraph Restriction)**.** *Given a multigraph $M = (V, E^+, <, nw, na)$ of multigraph type $MT = (D, W, L, dl, A, T, dat)$, we define its restriction to dimensions $D' \subseteq D$ as:*

$$M_{|D'} \quad = \quad (V, E^+_{|D'}, <, nw, na_{|D'})$$

*where $E^+_{|D'}$ is the set of edges restricted to $D'$:*

$$E^+_{|D'} \quad = \quad \{(v, v, l, d) \mid (v, v, l, d) \in E^+ \wedge\ d \in D'\}$$

*and $na_{|D'}$ is the node-attributes mapping restricted to $D'$, which we define as follows for all $v \in V$:*

$$na_{|D'}\ v \quad = \quad \{d \mapsto \{a \mapsto u \mid u \in na\ v\ d\ a\} \mid d \in D'\}$$

This brings us to the following lemma.

**Lemma 1** (Grammar Composition)**.** *Basic set operations on the string languages licensed by two XDG grammars $G_1$ and $G_2$ can be realized using grammar composition $G_1 \circledcirc G_2$ with the corresponding principles composition operator $\circ$.*

*Proof.* We start with:

$$L\ (G_1 \circledcirc G_2) \quad = \quad L\ G_1 \circledcirc L\ G_2$$

It follows that:

$$s \in L\ (G_1 \circledcirc G_2) \quad \equiv \quad s \in L\ G_1\ \circ\ s \in L\ G_2$$

where $\circ$ is the logical operator corresponding to the set operation $\circledcirc$. By Definition 10:

$$s \in \{s\ M \mid M \in m\ (G_1 \circledcirc G_2)\}\ \equiv$$
$$s \in \{s\ M \mid M \in m\ G_1\}\ \circ\ s \in \{s\ M \mid M \in m\ G_2\}$$

As the models of $G_1$ are determined by dimensions $D_1$, and those of $G_2$ by $D_2$, we know:

$$s \in \{s\ M_{|D_1 \cup D_2} \mid M_{|D_1 \cup D_2} \in m\ (G_1 \circledcirc G_2)\}\ \equiv$$
$$s \in \{s\ M_{|D_1} \mid M_{|D_1} \in m\ G_1\}\ \circ\ s \in \{M_{|D_2} \mid M_{|D_2} \in m\ G_2\}$$

From which it follows that:

$$M_{|D_1 \cup D_2} \in m\ (G_1 \circledcirc G_2)\ \equiv\ M_{|D_1} \in m\ G_1\ \circ\ M_{|D_2} \in m\ G_2$$

□

That is, a multigraph $M$ with dimensions $D_1 \cup D_2$ is a model in $m\ (G_1 \circledcirc G_2)$ if and only if $M$ restricted to $D_1$ is a model of $m\ G_1$, $M$ restricted to $D_2$ is a model of $m\ G_2$, and $M_{|D_1} \in m\ G_1\ \circ\ M_{|D_2} \in m\ G_2$ is true. That is, we can model the set operation $\circledcirc$ using grammar composition with the logical principles composition operator $\circ$.

## 3.2 Difference and Complement

Our first use of lemma 1 is to show that the XDG string languages are closed under difference.

**Proposition 1** (Difference)**.** *The XDG string languages are closed under difference.*

*Proof.* Given two grammars $G_1$ and $G_2$ with string languages $L\ G_1$ and $L\ G_2$, we can, by lemma 1, construct a grammar $G = G_1 - G_2$ with $L\ G = L\ G_1 - L\ G_2$ using grammar composition with principles composition operator $\circ = \lambda p_1.\lambda p_2.p_1 \wedge \neg p_2$:

$$P \quad = \quad \bigwedge_{\phi_1 \in P_1} \phi_1 \quad \wedge \quad \neg \bigwedge_{\phi_2 \in P_2} \phi_2$$

□

**Proposition 2** (Complement)**.** *The XDG string languages are closed under complement.*

*Proof.* Given a grammar $G_2$ with string language $L\ G_2$, lemma 1 allows us to construct a grammar $G = G_1 - G_2$ with $L\ G = L\ G_1 - L\ G_2 = \emptyset - L\ G_2 = \overline{G_2}$ using grammar composition with principles composition operator $\circ = \lambda p_1.\lambda p_2.p_1 \wedge \neg p_2$ and a "dummy grammar" $G_1$, whose string language is the empty set. □

### 3.3 Union

**Proposition 3** (Union)**.** *The XDG string languages are closed under union.*

*Proof.* Given two grammars $G_1$ and $G_2$ with string languages $L\ G_1$ and $L\ G_2$, we can construct a grammar $G = G_1 \cup G_2$ with $L\ G = L\ G_1 \cup L\ G_2$ using grammar composition with principles composition operator $\circ = \vee$:

$$P \quad = \quad \bigwedge_{\phi_1 \in P_1} \phi_1 \quad \vee \quad \bigwedge_{\phi_2 \in P_2} \phi_2$$

□

### 3.4 Intersection

**Proposition 4** (Intersection)**.** *The XDG string languages are closed under intersection.*

*Proof.* Given two grammars $G_1$ and $G_2$ with string languages $L\ G_1$ and $L\ G_2$, we can construct a grammar $G = G_1 \cap G_2$ with $L\ G = L\ G_1 \cap L\ G_2$ using grammar composition with principles composition operator $\circ = \wedge$:

$$P \quad = \quad \bigwedge_{\phi_1 \in P_1} \phi_1 \quad \wedge \quad \bigwedge_{\phi_2 \in P_2} \phi_2$$

□

### 3.5 Example

As an example, we present the intersection of the two grammars $G_1$ and $G_2$ from section 2 to obtain the language $L_3 = L_1 \cap L_2$ of *n a*s followed by *n b*s followed by *n c*s.

$$L_3 \quad = \quad L_1 \cap L_2 \quad = \quad \{s \in a^n b^n c^n \mid n \geq 1\}$$

The models of $G_3$ are multigraphs with two dimensions: the dimension ID from $G_1$, and the dimension LP from $G_2$. ID ensures that there are equally many *a*s, *b*s and *c*s, whereas LP ensures that the *a*s precede the *b*s precede the *c*s. We depict an example analysis in Figure 8.

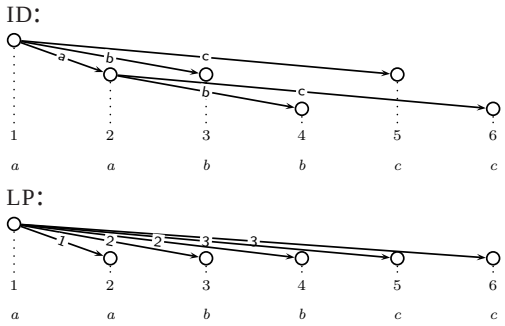



Figure 8: $G_3$ ID/LP example analysis of *a a b b c c*

The lexicon of $G_3$ is the product of the lexicons of $G_1$ and $G_2$. We depict it in Figure 9. Note that the product construction of the lexicon yields two lexical entries for *a* which are different on LP, but equal on ID.

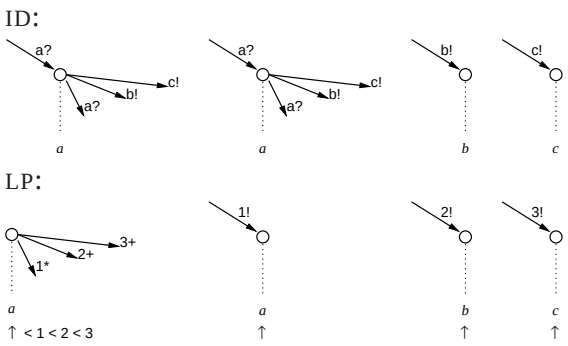



Figure 9: $G_3$ lexical entries for *a*, *b* and *c*

## 4 LCFGs as XDGs

(Debusmann, 2006) includes a constructive proof based on (McCawley, 1968) and (Gaifman, 1965) that reformulates lexicalized CFGs (LCFGs) as

XDGs. LCFGs are CFGs where each rule has precisely one terminal symbol on its right hand side. Given an LCFG $G$, it is easy to construct an XDG $G'$ with one dimension called DERI (for "derivation tree"). The derivation trees of the LCFG stand in the following correspondence to the models on DERI:

1. The non-terminal nodes in the derivation tree correspond to the nodes on DERI.

2. The labels of the non-terminal nodes in the derivation tree are represented by the incoming edge labels of the corresponding nodes on DERI, except for the root, which has no incoming edge.

3. The terminal nodes in the derivation tree correspond to the words on DERI.

We depict an example LCFG derivation tree and the corresponding XDG DERI tree in Figure 10.

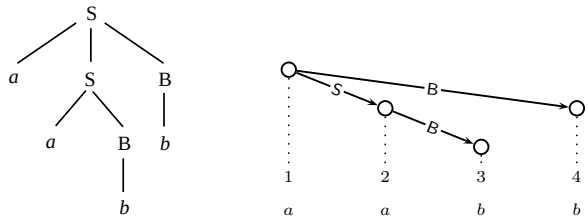



Figure 10: LCFG derivation tree (left) and corresponding XDG DERI tree (right)

The constructed XDG grammar uses the tree, projectivity, valency and order principles. The lexicon includes for each rule $A \to B_1 \dots B_k a B_{k+1} \dots B_n$ ($1 \leq k \leq n$) of the LCFG, given that each non-terminal occurs at most once on the RHS, and given that $A$ is not the start symbol, a lexical entry graphically depicted in Figure 11. Here, the anchor is the terminal symbol $a$ of the RHS of the LCFG rule. We require precisely one incoming edge labeled by the LHS of the rule, i.e., $A$.[2] As for the outgoing edges, we require precisely one for each non-terminal on the RHS of the rule. The order requirements reflect the order among the non-terminals and the anchor.

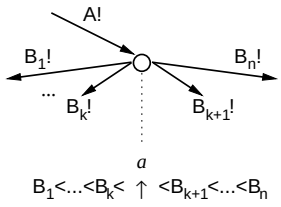



Figure 11: Lexical entry for LCFG rule $A \to B_1 \dots B_k a B_{k+1} \dots B_n$

# 5 Scrambling as the Combination of Relaxed LCFGs

In German, following the theory of *topological fields*, the word order in subordinate sentences is such that all verbs are positioned in the so-called *verb-cluster* at the right end, preceded by the non-verbal dependents (e.g. NPs) in the so-called *Mittelfeld*. In the verb cluster, the heads follow their dependents. We show an example in Figure 12, where the subscripts indicate the dependencies between the NPs and the verbs: *John* and *Mary* are dependents of *sah*, *Peter* of *helfen* and *Tiere* of *füttern*.

Figure 13 shows an LCFG called $G_{\text{ID}}$ which generates this word order. The problem with this grammar is that it generates only one analysis for the example sentence, shown in Figure 14 (left), whereas 12 are grammatical. This is because the NPs in the Mittelfeld can occur in any permutation[3] irrespec-

[2] If $A$ is the start symbol, we license zero or one incoming edges labeled $A$ instead of precisely one.

[3] Any permutation is *grammatical*, though some are strongly marked.

| Mittelfeld | | | | | verb cluster | | |
|---|---|---|---|---|---|---|---|
| *(dass)* | *John*$_1$ | *Mary*$_1$ | *Peter*$_2$ | *Tiere*$_3$ | *füttern*$_3$ | *helfen*$_2$ | *sah*$_1$ |
| *(that)* | *John*$_1$ | *Mary*$_1$ | *Peter*$_2$ | *animals*$_3$ | *feed*$_3$ | *help*$_2$ | *saw*$_1$ |

Figure 12: German subordinate clause version of the English sentence *(that) John saw Mary help Peter feed animals.*

tively of the positions of their verbal heads.[4] In order to correctly model this so-called *scrambling* phenomenon, we would also have to also license e.g. the discontinuous analysis shown in Figure 14 (middle). But how can we do that, given that LCFG derivations are always contiguous?

| | | | | | |
|---|---|---|---|---|---|
| S | → | NP NP VP *sah* | VP | → | NP VP *helfen* |
| VP | → | NP *füttern* | NP | → | *John* |
| NP | → | *Mary* | NP | → | *Peter* |
| NP | → | *Tiere* | | | |

Figure 13: LCFG $G_{\text{ID}}$

## 5.1 Relaxing LCFGs

Our first idea is to reformulate $G_{\text{ID}}$ in XDG. In XDG, we can then relax the global contiguity constraint by simply dropping the projectivity principle.

But this is not quite the solution as it leads to overgeneration: although the rules for VPs still position their verbal dependents to their left, material from verbs higher up in the tree can now interrupt them, as in Figure 14 (right), where the VP *Peter Tiere füttern helfen* is interrupted by the NPs *John* and *Mary*, and as a result, the verb *füttern* wrongly ends up in the Mittelfeld.

[4]Why 12? The verb *füttern* has 4 possibilities to fill its NP argument slot, there remain 3 for *helfen*, and 1 for *sah*.

## 5.2 Topological LCFG

Our second idea is to create a new, *topological* LCFG called $G_{\text{LP}}$ in the spirit of topological fields theory, as in (Kathol, 1995), (Gerdes and Kahane, 2001), (Duchier and Debusmann, 2001). $G_{\text{LP}}$ basically orders all NPs to the left of the verbs. We use the non-terminals MF standing for "Mittelfeld" and VC for "Verb Cluster". The grammar is depicted in Figure 15, and an example analysis in Figure 16.

| | | | | | |
|---|---|---|---|---|---|
| S | → | MF VC *sah* | VC | → | VC *helfen* |
| VC | → | *füttern* | MF | → | *John* |
| MF | → | *John* MF | MF | → | *Mary* |
| MF | → | *Mary* MF | MF | → | *Peter* |
| MF | → | *Peter* MF | MF | → | *Tiere* |
| MF | → | *Tiere* MF | | | |

Figure 15: Topological LCFG $G_{\text{LP}}$

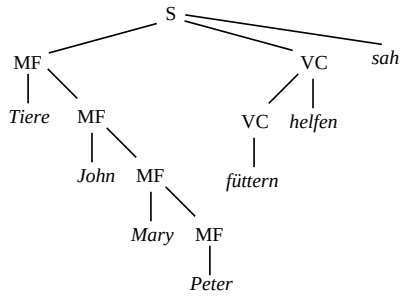



Figure 16: Topological derivation tree for *(dass) Tiere*$_3$ *John*$_1$ *Mary*$_1$ *Peter*$_2$ *füttern*$_3$ *helfen*$_2$ *sah*$_1$.

However, solely using the $G_{\text{LP}}$ is not viable: although we get precisely the correct string language, the derivation trees do not represent the syntactic dependencies between verbs and their non-verbal dependents, e.g. between *sah* and *John* and *Mary*. This renders the grammar practically useless: it is impossible to determine the semantics of a sentence without these syntactic dependencies.

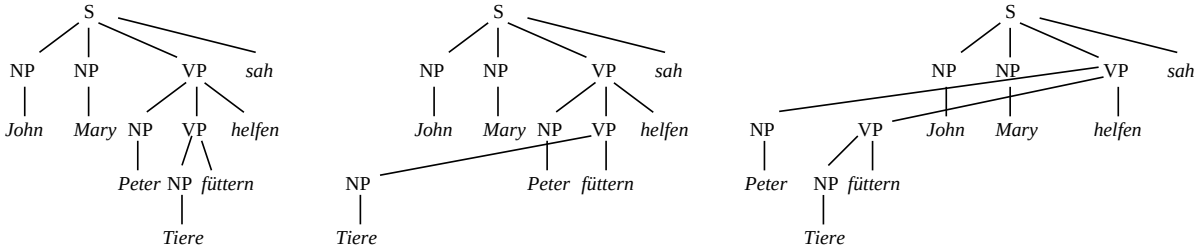


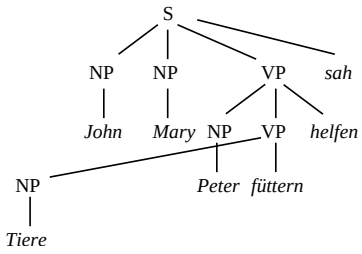

Figure 14: Derivation trees

### 5.3 Intersecting LCFGs

To recap our two previous ideas, relaxing $G_{\text{ID}}$ lead to overgeneration, and the sole use of the topological LCFG $G_{\text{LP}}$ made us lose essential syntactic dependencies. Our third idea is now to *intersect* $G_{\text{ID}}$ and $G_{\text{LP}}$, following section 3.4. An analysis of the resulting grammar $G_{\text{ID/LP}} = G_{\text{ID}} \cap G_{\text{LP}}$ is a pair of two derivation trees, or, in terms of XDG, two *dimensions*: one derivation tree for $G_{\text{ID}}$ called ID tree, and one derivation tree for $G_{\text{LP}}$ called LP tree. We show an example in Figure 17.

This idea combines the best of both worlds: through $G_{\text{LP}}$, we avoid overgeneration, and $G_{\text{ID}}$ represents the essential syntactic dependencies. That is, the two intersected grammars can be considered as "helping out" each other.

ID:




LP:

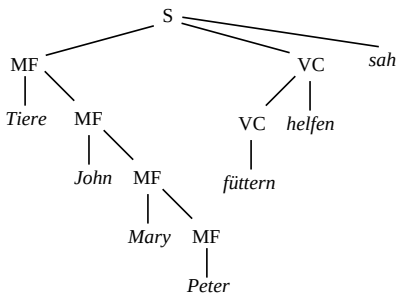



Figure 17: Analysis of $G_{\text{ID/LP}}$

## 6 Use or Abuse of Intersection?

A related approach to model scrambling by intersection has been put forward in the context of Range Concatenation Grammars (RCG) (Boullier, 2000). Here, the structures generated by the two combined grammars are correlated only by their yields. In his paper "Uses and abuses of intersected languages", Chiang (2004) observes that from the point of view of strong generative capacity, this use of intersection amounts to only constraining the tail end of otherwise independent parallel processes, which he calls *weak parallelism*. He argues that it is easy to overestimate how much control this kind of parallelism offers. He argues that the treatment of scrambling in (Boullier, 2000) is not general enough, as it relies on nonexistent information in the surface string.

Intersection in XDG offers more fine-grained control as Boullier's, and we argue that it thus does not fall into the category of "abuse". First, the dimensions of XDG are synchronized by the input string and the corresponding nodes, which are shared among all dimensions. Second, XDG allows to stipulate any number of additional constraints to correlate the two intersected grammars, such as the linking principle. Linking constraints could e.g. be used to synchronize the rules of the two combined CFGs. For instance, we could use it to require that specific rules in one of the combined CFGs can only be used synchronously with specific rules in the other CFG, similar to Multitext grammars (Melamed, 2003), (Melamed et al., 2004).

## 7 Conclusions

We have shown that XDGs can be *combined* using *grammar composition*, such that the string language of the resulting grammar is e.g. their *intersection*. Using a model-theoretic axiomatization of LCFG in XDG, we could then explore the *relaxation* of the LCFG contiguity criterion, and, crucially, the *intersection* of LCFGs. Together, these two ideas lead us to a model of one of the most complicated phenomena in syntax by the combination of two grammars formulated in one of the simplest of all grammar formalisms.

## Acknowledgments

I'd like to thank Prof. Gert Smolka from Programming Systems Lab in Saarbrücken, the people from the CHORUS project, and the International Graduate College (IGK) Saarbrücken/Edinburgh for supporting my research over the years. I'd also like to thank the anonymous reviewers of this paper for their valuable suggestions.

## References

Apt, Krzysztof R. (2003). *Principles of Constraint Programming*. Cambridge University Press.

Blackburn, Patrick and Claire Gardent (1995). A specification language for Lexical Functional Grammars. In *Proceedings of EACL 1995*. Dublin/IE.

Boullier, Pierre (2000). Range Concatenation Grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pp. 53–64. Trento/IT.

Chiang, David (2004). Uses and abuses of intersected languages. In *Proceedings of TAG+7*, pp. 9–15. Vancouver/CA.

Debusmann, Ralph (2006). *Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description*. Ph.D. thesis, Universität des Saarlandes.

Debusmann, Ralph (2007). The complexity of First-Order Extensible Dependency Grammar. Technical report, Saarland University.

Debusmann, Ralph and Denys Duchier (2007). XDG Development Kit. Http://www.mozart-oz.org/mogul/info/debusmann/xdk.html.

Debusmann, Ralph, Denys Duchier, and Joachim Niehren (2004). The XDG grammar development kit. In *Proceedings of the MOZ04 Conference*, volume 3389 of *Lecture Notes in Computer Science*, pp. 190–201. Springer, Charleroi/BE.

Duchier, Denys and Ralph Debusmann (2001). Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of ACL 2001*. Toulouse/FR.

Gaifman, Haim (1965). Dependency systems and phrase-structure systems. *Information and Control*, **8**(3):304–337.

Gerdes, Kim and Sylvain Kahane (2001). Word order in German: A formal dependency grammar using a topological hierarchy. In *ACL 2001 Proceedings*. Toulouse/FR.

Hudson, Richard A. (1990). *English Word Grammar*. B. Blackwell, Oxford/UK.

Kathol, Andreas (1995). *Linearization-Based German Syntax*. Ph.D. thesis, Ohio State University, Ohio/US.

Kepser, Stephan (2000). A coalgebraic modelling of Head-driven Phrase Structure Grammar. In *Proceedings of AMiLP 2000*.

Kepser, Stephan and Uwe Mönnich (2003). Graph properties of HPSG feature structures. In Gerhard Jäger, Paola Monachesi, Gerald Penn, and Shuly Wintner, eds., *Proceedings of Formal Grammar 2003*, pp. 115–124.

Kracht, Marcus (1995). Syntactic codes and grammar refinement. *Journal of Language, Logic and Information*, **4**:41–60.

McCawley, J. D. (1968). Concerning the base component of a Transformational Grammar. *Foundations of Language*, **4**:243–269.

Melamed, I. Dan (2003). Multitext grammars and synchronous parsers. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, pp. 158–165. Edmonton/CA.

Melamed, I. Dan, Giorgio Satta, and Benjamin Wellington (2004). Generalized Multitext Grammars. In *Proceedings of ACL 2004*. Barcelona/ES.

Pullum, Geoffrey K. and Barbara C. Scholz (2001). On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retoré, eds., *Logical Aspect of Computational Linguistics: 4th International Conference*, Lecture Notes in Artificial Intelligence, pp. 17–43. Springer, Berlin/DE.

Rogers, James (1996). A model-theoretic framework for theories of syntax. In *Proceedings of ACL 1996*.

Rogers, James (2003). Syntactic structures as multidimensional trees. *Journal of Research on Language and Computation*, **1**(3/4).

Rogers, James (2004). On scrambling, another perspective. In *Proceedings of TAG+7*. Vancouver/CA.

Schulte, Christian (2002). *Programming Constraint Services*, volume 2302 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Vardi, Moshe Y. (1982). The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of Computing*, pp. 137–146. ACM Press, San Francisco/US.