

# 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 in MTS has still been invested in reformulations of GES frameworks: of GB 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 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This paper contributes a new, previously unpublished formalization of XDG in first-order logic (section 2), and the first results on the closure properties of the string languages licensed by XDG (section 3).

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 sets of dependency graphs sharing the same set of nodes, which are anchored by the same string of words. The individual dependency graphs are called *dimensions*, and entire XDG analyses are called *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*.
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

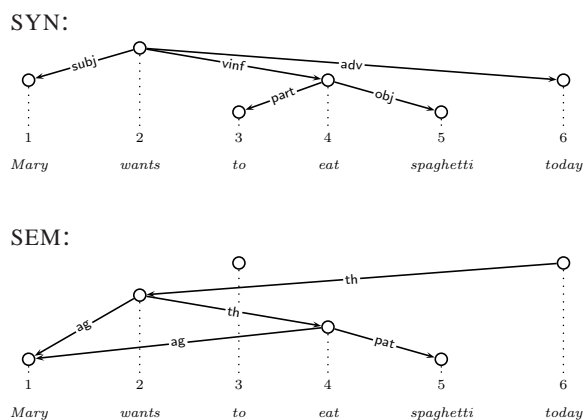


Figure 1: XDG multigraph for *Mary wants to eat spaghetti today*.

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*, arbitrary 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.

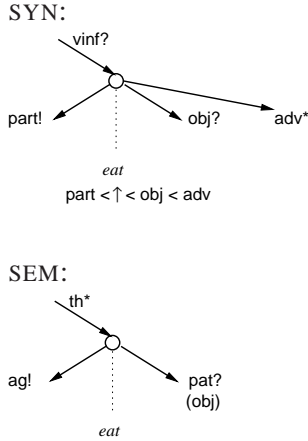


Figure 2: Lexical entry for the word *eat*

## 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 set of atoms  $At$ , a finite set of edge labels  $L \subseteq At$ , a finite set of dimensions  $D \subseteq At$ , a finite set of words  $W \subseteq At$ , a finite set of attributes  $A \subseteq At$ , a finite set of set types  $T$ , and a set of values  $U = \cup\{t \mid t \in T\}$ , a multigraph  $M = (V, E^+, <, nw, na)$  consists of a finite set of nodes  $V$ ,*

*a set of labeled dominances  $E^+ \subseteq V \times V \times L \times D$ , a strict total order  $< \subseteq V \times V$  on  $V$ , a node-word mapping  $nw \in V \rightarrow W$ , and a node-attributes mapping  $na \in V \rightarrow D \rightarrow A \rightarrow U$ . We define  $V$  as a finite interval of the natural numbers starting with 1. A labeled dominance  $(v, v', l, d)$  is an element of  $E^+$  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. Each value  $u \in U$  is an element of a set type  $t \in T$ , where  $t = 2^{Fd_1 \times \dots \times Fd_n}$  and  $Fd_i \subseteq At$ . That is, each value is a set of tuples whose components are atoms from finite domains.*

## 2.2 Grammar

**Definition 2** (Grammar). *A grammar  $G = (MT, lex_{MT, A'}, P_{MT})$  consists of a multigraph type  $MT$ , a lexicon  $lex_{MT, A'}$ , and a set of principles  $P_{MT}$ . The lexicon  $lex_{MT, A'}$  is defined over multigraph type  $MT$  and a subset  $A' \subseteq A$  of the attributes called *lexical attributes*. The principles  $P_{MT}$  are defined over the same multigraph type  $MT$ . We will drop the subscripts whenever this is convenient.*

**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  $dl \in D \rightarrow 2^L$ , a finite set of attributes  $A \subseteq At$ , a finite set of types  $T$ , and a dimension-attributes-type mapping  $dat \in D \rightarrow A \rightarrow T$ . The dimension-label mapping determines which labels can be used on which dimension, and the dimension-attributes-type mapping determines values of which type can be used for which attribute on which dimension. Each  $t \in T$  is a set type  $2^{Fd_1 \times \dots \times Fd_n}$ , where  $Fd_i \subseteq At$ . Each multigraph type induces the set  $U = \cup\{t \mid t \in T\}$  of values.*

**Definition 4** (Multigraph of Multigraph Type). *A*

multigraph  $M = (V, E^+, <, nw, na)$ , defined over the sets  $L'$  of edge labels,  $D'$  of dimensions,  $W'$  of words,  $A'$  of attributes, and  $T'$  of types is of multigraph type  $MT = (D, W, L, dl, A, T, dat)$  iff  $L' \subseteq L$ ,  $D' = D$ ,  $W' \subseteq W$ ,  $A' = A$  and  $T' = T$ , all labeled dominances on dimension  $d \in D'$  have only edge labels in  $dl\ d$ , and all node attributes  $a \in A'$  on dimension  $d \in D'$  are properly typed, i.e., have a value in  $dat\ d\ a$ .

**Definition 5** (Lexicon). The lexicon  $lex_{MT, A'}$  is defined over a multigraph type  $MT = (D, W, L, dl, A, T, dat)$  and a subset  $A' \subseteq A$  of the attributes called lexical attributes. It is a function from words to sets of lexical entries:  $lex_{MT, A'} \in W \rightarrow 2^{D \rightarrow A' \rightarrow U}$ , where for all  $w \in W$ , if  $e \in lex\ w$ , then for all  $d \in D$ ,  $a \in A'$ ,  $(e\ d\ a)$  is properly typed, i.e., has a value in  $(dat\ d\ a)$ .

**Definition 6** (Principles). The principles  $P_{MT}$  are defined over a multigraph type  $MT = (D, W, L, dl, A, T, dat)$ . They are a finite set  $P_{MT} \subseteq \phi$  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:

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

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  $C$ :

$$C = DUWULUAUFUN$$

where  $F = \cup\{Fd_1 \cup \dots \cup Fd_n \mid 2^{Fd_1 \times \dots \times Fd_n} \in T\}$  and  $\mathbb{N}$  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. The predicates  $\Psi$  are defined as follows:

$$\begin{aligned} \Psi ::= & v < v' \\ & \mid v \xrightarrow{l}_d \rightarrow_d^* v' \\ & \mid w(v) = w \\ & \mid (t_1 \dots t_n) \in a_d(v) \end{aligned}$$

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 strict total order  $<$ , i.e.,  $(v, v') \in <$ .  $w(v) = w$  is interpreted by the node-word mapping, i.e.,  $nw\ v = w$ , and  $(t_1 \dots t_n) \in a_d(v)$  by the node-attributes mapping, i.e.,  $(t_1, \dots, t_n) \in na\ v\ d\ a$ .

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

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

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  $G = (MT, lex, P)$ ,  $m \in G$ , 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  $\bigwedge_{\phi \in P} \phi$  of all principles in  $P$  is true.*

## 2.4 String Language

To arrive at the string language of an XDG grammar, we first define the yield of a multigraph.

**Definition 10** (Yield of a Multigraph). *The yield of a multigraph  $M = (V, E^+, <, nw, na)$  is the concatenation of the words of the nodes, ordered with respect to the strict total order  $<$ :*

$$y\ M = nw\ p_i \dots nw\ p_{|V|}$$

where for all  $i, j, 1 \leq i < j \leq |V|, (p_i, p_j) \in <$ .

**Definition 11** (String Language). *The string language  $L\ G$  of a grammar  $G$  is the set of yields of the models of  $G$ :*

$$L\ G = \{y\ M \mid M \in m\ G\}$$

## 2.5 Recognition

**Definition 12** (XDG Recognition Problem). *Given a grammar  $G$  and a string  $s$ , is  $s$  in  $L\ G$ ?*

We are given a grammar  $G = (MT, lex, P)$  and an input string  $s = a_1 \dots a_n$ . We need to find a multigraph  $M = (V, E^+, <, nw, na)$  of multigraph type  $MT$  where:

1.  $V = \{1, \dots, n\}$
2.  $< = \{(i, j) \mid i < j\}$
3.  $nw = \{i \mapsto a_i \mid 1 \leq i \leq n\}$

As the only sets which are not determined by the input string,  $E^+$  and  $na$ , are finite, we can enumerate all multigraphs which satisfy the criteria 1., 2. and 3. above. If among them we find one that is a model of  $G$ , then  $s \in L\ G$ , if not, then  $s \notin L\ G$ .

In (Debusmann, 2007), we prove the complexities of three flavors of the recognition problem using results from (Vardi, 1982):

1. Universal Recognition Problem where both  $G$  and  $s$  are variable: *PSPACE-complete*.
2. Fixed Recognition Problem where  $G$  is fixed and  $s$  is variable: *NP-complete*.
3. Instance Recognition Problem where the principles are fixed, and the lexicon and  $s$  are variable: also *NP-complete*.

## 2.6 Parsing

**Definition 13** (XDG Parsing Problem). *Given a grammar  $G$  and a string  $s$ , find all models  $M \in m\ G$  where  $y\ M = s$ .*

Again we are given a grammar  $G = (MT, lex, P)$  and an input string  $s = a_1 \dots a_n$ . The parsing problem is finding all multigraphs  $M = (V, E^+, <, nw, na) \in m\ G$  where:

1.  $V = \{1, \dots, n\}$
2.  $< = \{(i, j) \mid i < j\}$
3.  $nw = \{i \mapsto a_i \mid 1 \leq i \leq n\}$

That is, the set of nodes, the strict total order  $<$  and the node-word mapping  $nw$  are determined by the input string. Parsing then consists of simply a) adding a finite number of edges between these nodes and b) finding an appropriate node-attributes mapping. Crucially, no nodes need to be 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).

## 2.7 Generation

**Definition 14** (XDG Generation Problem). *Given a grammar  $G$  and a bag of words  $b$ , find all models  $M \in mG$  where  $yM = s$  and  $s$  is a linearization of  $b$ .*

We are given a grammar  $G = (MT, lex, P)$  and an input bag of words  $b = \{a_1, \dots, a_n\}$ . The generation problem is finding all multigraphs  $M = (V, E^+, <, nw, na) \in mG$  where:

1.  $V = \{1, \dots, n\}$
2.  $nw = \{i \mapsto a_i \mid 1 \leq i \leq n\}$

Thus, generation consists of a) adding a finite a finite number of edges between the nodes, b) finding an appropriate node-attributes mapping, and, in addition to parsing, c) finding an appropriate strict total order on the set of nodes.

## 2.8 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:

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

**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.

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

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  $(dl\ d) \rightarrow \{!, +, ?, *\}$  from edge labels on  $d$  to *cardinalities*, where ! stands for “one”, + for “more than one”, ? for “zero or one”, and \* for “ar-

$$eat \mapsto \left( \left( \left( \begin{array}{l} \text{SYN} : \left\{ \begin{array}{l} in : \{(vinf, ?)\} \\ out : \{(part, !), (obj, ?), (adv, *)\} \\ order : \{(part, \uparrow), (part, obj), \\ (part, adv), (\uparrow, obj), \\ (\uparrow, adv), (obj, adv)\} \end{array} \right\} \\ \text{SEM} : \left\{ \begin{array}{l} in : \{(th, *)\} \\ out : \{(ag, !), (pat, ?)\} \\ link : \{(pat, obj)\} \end{array} \right\} \end{array} \right) \right) , \dots \right)$$

Figure 3: Concrete lexical entry for *eat*

bitrary many”.

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

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  $\uparrow$ .

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

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  $l' \in link_{d_1}(v)$ , 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  $(dl\ d_1) \rightarrow 2^{(dl\ d_2)}$  mapping labels on  $d_1$  to sets of labels on  $d_2$ .

$$\begin{aligned} linking_{d_1, d_2} = \\ \forall v : \forall v' : \forall l : \forall l' : \\ v \xrightarrow{l}_{d_1} v' \wedge (l, l') \in link_{d_1}(v) \Rightarrow v \xrightarrow{l'}_{d_2} v' \end{aligned}$$

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,  $d_1 = \text{SEM}$  and  $d_2 = \text{SYN}$ , and the linking principle stipulates e.g. that the patient of *eat* on SEM must be realized by its object on SYN.

## 2.9 Example Grammars

To illustrate how XDG grammars look like, we present two example grammars. The first,  $G_1$ , models the string language  $L_1$  of equally many *as*, *bs* and *cs*, in any order:

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

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 *as*, 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.

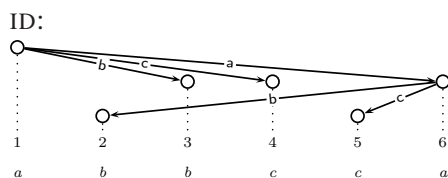


Figure 4:  $G_1$  ID example analysis of *a b b c c a*

$G_1$  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 *as* 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*.

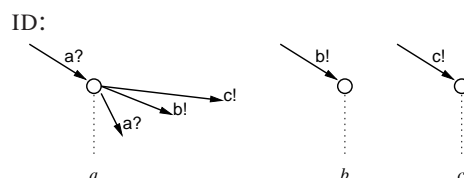


Figure 5:  $G_1$  lexical entries for *a*, *b* and *c*

The second example grammar,  $G_2$ , models the string language  $L_2$  of arbitrary many *as* followed by arbitrary many *bs* followed by arbitrary many *cs*:

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

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 *as* (labeled 1), and *bs* (2) and *cs* (3) to its right. We show an example analysis in Figure 6.

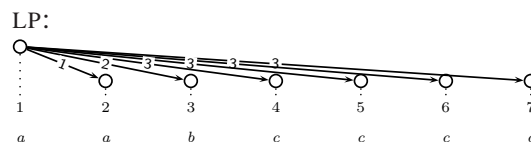


Figure 6:  $G_2$  LP example analysis of *a a b b c c*

$G_2$  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 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.



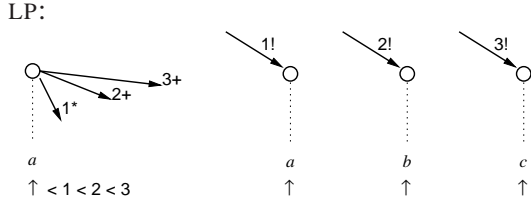


Figure 7:  $G_2$  lexical entries for  $a$ ,  $b$  and  $c$

### 3 Closure Properties

In this section, we will present our first results on the closure properties of XDG. We will see that the string languages licensed by XDG are closed under intersection and union.

#### 3.1 Multigraph Restriction

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

**Definition 15** (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:*

$$M_{|D'} = (V, E_{|D'}^+, <, nw, na_{|D'})$$

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

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

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

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

#### 3.2 Intersection

We first define the notion of *grammar intersection*, i.e., the combination of two grammars  $G_1$  and  $G_2$  to

a new grammar  $G$  such that the set of models of  $G$  is the intersection of the set of models of  $G_1$  and the set of models of  $G_2$ .

**Definition 16** (Grammar Intersection). *We define grammar intersection given the following two grammars:*

$$\begin{aligned} G_1 &= (MT_1, lex_1, P_1) \\ G_2 &= (MT_2, lex_2, P_2) \end{aligned}$$

where:

$$\begin{aligned} MT_1 &= (D_1, W_1, L_1, dl_1, A_1, T_1, dat_1) \\ MT_2 &= (D_2, W_2, L_2, dl_2, A_2, T_2, dat_2) \end{aligned}$$

We write  $G = G_1 \cap G_2$  for the combined grammar  $G = (MT, lex, P)$ . For the set of models  $m G$ , it holds that:

$$m(G_1 \cap G_2) = m G_1 \cap m G_2$$

and, equivalently, for all multigraphs  $M$ :

$$M \in m(G_1 \cap G_2) \equiv M \in m G_1 \wedge M \in m G_2 \quad (1)$$

The preconditions for grammar intersection are:

1. the sets of dimensions must be disjoint:  $D_1 \cap D_2 = \emptyset$
2. the sets of words must be equal:  $W_1 = W_2$

The multigraph type  $MT = (D, W, L, dl, A, T, dat)$  of  $G$  is defined as follows:

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

The lexicon  $lex$  of  $G$  is defined such that for each word  $w \in W$ ,  $lex w$  contains the product of the lexical

entries for  $w$  from  $G_1$  and  $G_2$ :<sup>1</sup>

$$\text{lex } w = \{e_1 \cup e_2 \mid e_1 \in \text{lex}_1 w \wedge e_2 \in \text{lex}_2 w\}$$

The principles  $P$  of  $G$  conjoin the principles of  $G_1$  and  $G_2$ :

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

We can now proceed to prove that the string languages licensed by XDG are closed under intersection.

*Proof.* Consider the two grammars:

$$\begin{aligned} G_1 &= (MT_1, \text{lex}_1, P_1) \\ G_2 &= (MT_2, \text{lex}_2, P_2) \end{aligned}$$

where:

$$\begin{aligned} MT_1 &= (D_1, W_1, L_1, dl_1, A_1, T_1, dat_1) \\ MT_2 &= (D_2, W_2, L_2, dl_2, A_2, T_2, dat_2) \end{aligned}$$

and  $D_1 \cap D_2 = \emptyset$  and  $W_1 = W_2$ . The intersection of their string languages is:

$$L G_1 \cap L G_2$$

By Definition 11, this is equivalent to:

$$\{y M \mid M \in m G_1\} \cap \{y M \mid M \in m G_2\}$$

As  $G_1$  is restricted to  $D_1$  and  $G_2$  to  $D_2$ , we get the following using Definition 15:

$$\{y M_{|D_1} \mid M_{|D_1} \in m G_1\} \cap \{y M_{|D_2} \mid M_{|D_2} \in m G_2\}$$

Since  $G_1$  does not make use of the dimensions  $D_2$ , and  $G_2$  does not make use of  $D_1$ , we can safely write:

$$\{y M_{|D_1 \cup D_2} \mid M_{|D_1 \cup D_2} \in m G_1\} \cap \{y M_{|D_1 \cup D_2} \mid M_{|D_1 \cup D_2} \in m G_2\}$$

<sup>1</sup>This clarifies why we demand that  $G_1$  and  $G_2$  have the same set of words—otherwise, parts of the lexicon of  $G$  could not be defined.

which is equivalent to:

$$\{y M_{|D_1 \cup D_2} \mid M_{|D_1 \cup D_2} \in m G_1 \wedge M_{|D_1 \cup D_2} \in m G_2\}$$

which is in turn equivalent to:

$$\{y M \mid M \in m G_1 \wedge M \in m G_2\}$$

Using equation (1) from Definition 16, we get:

$$\{y M \mid M \in m (G_1 \cap G_2)\}$$

That is, we have that with grammar intersection, we can obtain the intersection of the string languages of  $G_1$  and  $G_2$ :

$$L G_1 \cap L G_2 \equiv \{y M \mid M \in m (G_1 \cap G_2)\}$$

□

### 3.3 Union

First, we define *grammar union*, analogously to grammar intersection.

**Definition 17** (Grammar Union). *We write  $G = G_1 \cup G_2$  for the combination of  $G_1$  and  $G_2$  where:*

$$m (G_1 \cup G_2) = m G_1 \cup m G_2$$

and, equivalently, for all multigraphs  $M$ :

$$M \in m (G_1 \cup G_2) \equiv M \in m G_1 \vee M \in m G_2 \quad (2)$$

*Grammar union is defined analogously to grammar intersection (Definition 16). The only difference concerns the definition of the principles  $P$  of  $G$ , which are combined disjunctively instead of conjunctively:*

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

The proof that the string languages licensed by XDG are closed under union then proceeds analogously to that for intersection.

### 3.4 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  $n$  *as* followed by  $n$  *bs* followed by  $n$  *cs*.

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

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 *as*, *bs* and *cs*, whereas LP ensures that the *as* precede the *bs* precede the *cs*. We depict an example analysis in Figure 8.

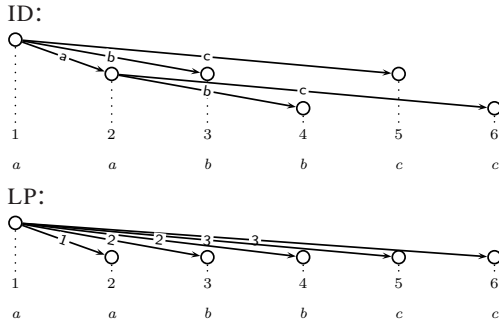


Figure 8:  $G_3$  ID/LP example analysis of  $a a b b c c$

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.

## 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.

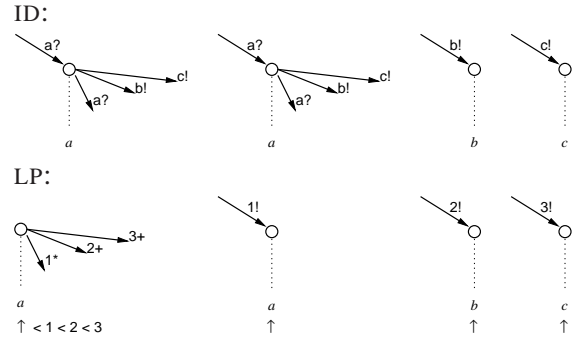


Figure 9:  $G_3$  lexical entries for *a*, *b* and *c*

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.

The constructed XDG grammar uses the tree, projectivity, valency and order principles. The lexicon includes for each rule  $A \rightarrow B_1 \dots B_k a B_{k+1} \dots B_n$  ( $1 \leq k \leq n$ ) 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,

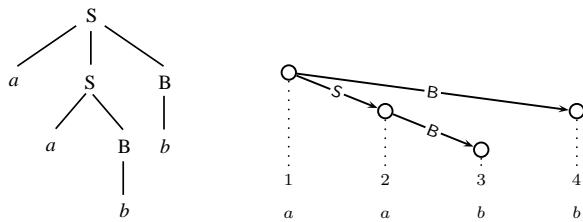


Figure 10: LCFG derivation tree (left) and corresponding XDG DERI tree (right)

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.

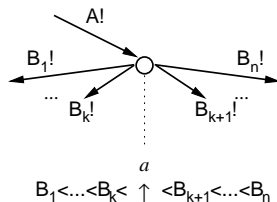


Figure 11: Lexical entry for LCFG rule  $A \rightarrow B_1 \dots B_k a B_{k+1} \dots B_n$

## 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

<sup>2</sup>If  $A$  is the start symbol, we license zero or one incoming edges labeled  $A$  instead of precisely one.

NPs and the verbs: *John* and *Mary* are dependents of *sah*, *Peter* of *helfen* and *Tiere* of *füttern*.

| Mittelfeld  | verb cluster                   |
|---|--------------------------------|
| ( <i>dass</i> ) $John_1$ $Mary_1$ $Peter_2$ $Tiere_3$   | $füttern_3$ $helfen_2$ $sah_1$ |
| ( <i>that</i> ) $John_1$ $Mary_1$ $Peter_2$ $animals_3$ | $feed_3$ $help_2$ $saw_1$      |

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

Figure 13 shows an LCFG called  $G_{ID}$  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<sup>3</sup> irrespectively of the positions of their verbal heads.<sup>4</sup> 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  | $\rightarrow$ | NP NP VP <i>sah</i> | VP | $\rightarrow$ | NP VP <i>helfen</i> |
| VP | $\rightarrow$ | NP <i>füttern</i>   | NP | $\rightarrow$ | <i>John</i>         |
| NP | $\rightarrow$ | <i>Mary</i>         | NP | $\rightarrow$ | <i>Peter</i>        |
| NP | $\rightarrow$ | <i>Tiere</i>        |    |               |                     |

Figure 13: LCFG  $G_{ID}$

<sup>3</sup>Any permutation is *grammatical*, though some are strongly marked.

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

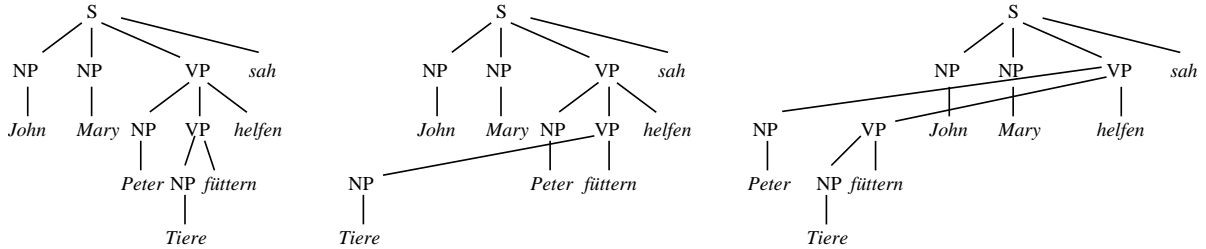


Figure 14: Derivation trees

### 5.1 Relaxing LCFGs

Our first idea is to reformulate  $G_{ID}$  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 over-generation: 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.

|    |   |           |    |   |           |
|----|---|-----------|----|---|-----------|
| 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  $G_{LP}$

### 5.2 Topological LCFG

Our second idea is to create a new, *topological* LCFG called  $G_{LP}$  in the spirit of topological fields theory, as in (Kathol, 1995), (Gerdes and Kahane, 2001), (Duchier and Debusmann, 2001).  $G_{LP}$  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.

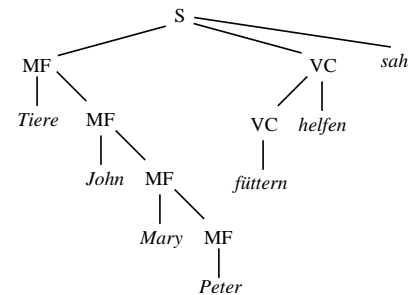


Figure 16: Topological derivation tree for (*dass*) *Tiere*<sub>3</sub> *John*<sub>1</sub> *Mary*<sub>1</sub> *Peter*<sub>2</sub> *füttern*<sub>3</sub> *helfen*<sub>2</sub> *sah*<sub>1</sub>.

However, solely using the  $G_{LP}$  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.

### 5.3 Intersecting LCFGs

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

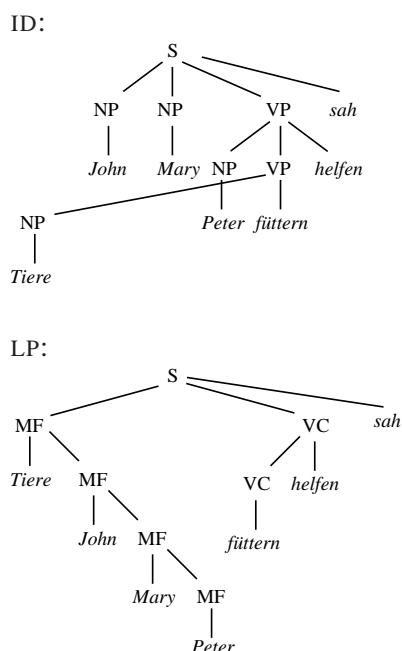


Figure 17: Analysis of  $G_{ID/LP}$

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

## 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 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 as 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](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, 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 multi-dimensional 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.