The Complexity of First-Order Extensible Dependency Grammar

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

Abstract

We rephrase the meta grammar formalism of Extensible Dependency Grammar (XDG) (Debusmann 2006), so far formalized in higher-order logic, in terms of first-order logic, and fill numerous gaps in the research on its complexity. In particular, there were no upper bounds of the complexity of the universal and fixed recognition problems. We prove that the universal recognition problem is PSPACE-complete, and the fixed recognition problem NP-complete. We introduce a new, practically relevant version of the universal recognition problem for *instances* of XDG, where only the lexicon is variable, and prove that it also is NP-complete.

1 Introduction

Extensible Dependency Grammar (XDG) (Debusmann 2006) is a meta grammar formalism combining ideas from dependency grammar (Tesnière 1959), model-theoretic syntax (Rogers 1996), and the parallel grammar architecture (Jackendoff 2002). In XDG, analyses are modularized into multiple levels of linguistic representation. This modular architecture is utilized e.g. in (Duchier & Debusmann 2001) for an elegant account of German word order phenomena such as scrambling, in (Debusmann, Duchier, Koller, Kuhlmann, Smolka & Thater 2004) for a relational syntax-semantics interface, and in (Debusmann, Postolache & Traat 2005) for a modular version of the prosodic account of information structure of Steedman (2000). XDG has also been used for efficient TAG-based generation in (Koller & Striegnitz 2002) and for parsing Polarized Unification Grammars in (Lison 2006).

Research on XDG has so far focused on practical aspects such as the the modeling of linguistic phenomena and the constraint parser implementation. Even though XDG has recently been formalized in higher-order logic (HOL) (Debusmann 2006), not much is known about its formal properties, e.g. its complexity:

1. Debusmann (2006) proves that the fixed recognition problem of XDG is NP-hard. But what is the upper bound of its complexity?

2. It follows that the universal recognition problem is also NP-hard. But how tight is this lower bound? And what is its upper bound?

3. As a meta grammar formalism, we are interested in the complexity of *instances* of XDG. But what is the corresponding recognition problem, and what is its complexity?

In this paper, we present a new formalization of XDG in terms of First-Order Logic (FOL) which enables us to answer these questions: we prove that the upper bound of the fixed recognition problem is in NP (question 1), the universal recognition problem PSPACE-complete (question 2), and we introduce a variant of the universal recognition problem for instances of XDG, proving that it is also NP-complete (question 3). The results are summed up in Table 1.

		(Debusmann 2006) (HOL)	this paper (FOL)
fixed RP	lower bound	NP-hard	NP-hard
	upper bound	?	in NP
universal RP	lower bound	NP-hard	PSPACE-hard
	upper bound	?	in PSPACE
instance RP	lower bound	?	NP-hard
	upper bound	?	in NP

Table 1: Complexity results for XDG recognition problems (RPs) in (Debusmann 2006) and this paper

2 Extensible Dependency Grammar

XDG is a description language for 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 analysis, 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.

In XDG, 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* stating the well-formedness conditions.

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

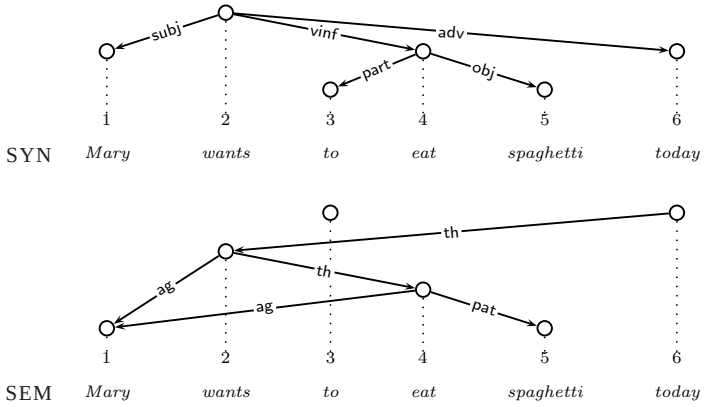
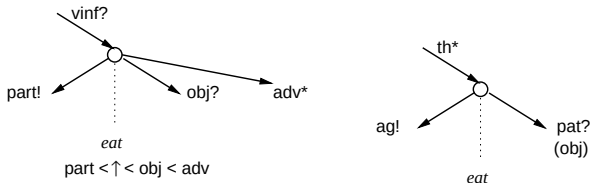


Figure 1: XDG multigraph for *Mary wants to eat spaghetti today*.

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. Below, we show an example schematic lexical entry for the word *eat*:



(1)

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 licenses no other 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, and licenses no other edges (valency principle). The patient must be realized an object (linking principle), the realization of the agent is not constrained.

3 First-Order Extensible Dependency Grammar

The main innovation of this paper is a new formalization of XDG called FO XDG, stated in FOL instead of HOL. This does not sacrifice the desired expressivity, as the principles defined in earlier papers on XDG and (Debusmann 2006) all only use first-order quantification.

3.1 Multigraph

We define multigraphs over the *labeled dominance relation* corresponding to the transitive closure of the labeled edge relation, where the label of the first edge is given. Only the inclusion of this relation allows us to stay in FOL: 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 $nw \in V \to W$, and a node-attributes mapping $na \in V \to D \to A \to U$. 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 $t = 2^{Fd_1 \times \ldots \times Fd_n}$ and $Fd_i \subseteq At$. That is, each value is a set of tuples whose components are atoms from finite domains.*

3.2 Grammar

Definition 2 (Grammar) *A grammar $G = (MT, lex_{MT,A'}, P_{MT})$ is made up 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 $dl \in D \to 2^L$, a finite set of attributes $A \subseteq At$, a finite set of types T, and a dimension-attributes-type mapping $dat \in D \to A \to T$. 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 $2^{Fd_1 \times \ldots \times Fd_n}$, 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 $M = (V, E^+, <, nw, na)$ which is 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: $lex_{MT,A'} \in W \to 2^{D \to A' \to U}$, 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. ϕ 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 *ψ 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:*

$$C \quad = \quad D \cup W \cup L \cup A \cup F \cup \mathbb{N}$$

where $F = \cup\{Fd_1 \cup \ldots \cup Fd_n \mid 2^{Fd_1 \times \ldots \times Fd_n} \in T\}$ 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 ψ are defined as follows:

$$\begin{array}{rcl} \psi & ::= & v < v' \\ & \mid & v \xrightarrow{l}_d \rightarrow_d^* v' \\ & \mid & w(v) = w \\ & \mid & (t_1 \ldots t_n) \in a_d(v) \end{array}$$

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 $(t_1 \ldots t_n) \in a_d(v)$ 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):

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

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

3.3 Example Principles

To get a deeper understanding of the principles of FO XDG, we show some representative example principles. For generality, the principles are parametrized by the dimensions that they constrain.

Definition 7 (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:

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

Definition 8 (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.*

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

For example, this principle is violated 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* which models the schematic lexical entry in (1) above. We write the lexical entry as a feature structure representing one point in the lexical mapping:

$$
\mathit{eat} \mapsto \left\{ \left\{ \begin{array}{ll}
\textsc{syn}: & \left\{ \begin{array}{l} \mathit{in} : \{(\mathsf{vinf}, ?)\} \\ \mathit{out} : \{(\mathsf{adv}, *), (\mathsf{obj}, ?), (\mathsf{part}, !)\} \\ \mathit{order} : \{(\mathsf{part}, \uparrow), (\mathsf{part}, \mathsf{obj}), (\mathsf{part}, \mathsf{adv}), (\uparrow, \mathsf{obj}), (\uparrow, \mathsf{adv}), (\mathsf{obj}, \mathsf{adv})\} \end{array} \right\} \\
\textsc{sem}: & \left\{ \begin{array}{l} \mathit{in} : \{(\mathsf{th}, *)\} \\ \mathit{out} : \{(\mathsf{ag}, !), (\mathsf{pat}, ?)\} \\ \mathit{link} : \{(\mathsf{pat}, \mathsf{obj})\} \end{array} \right\}
\end{array} \right\}, \ \ldots \right\} \tag{2}
$$

Definition 9 (Valency principle) *Given a dimension d, the purpose of the valency principle is to constrain 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 "arbitrary many".*

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

The remaining part of the principle dealing with the outgoing edges proceeds analogously.

Given the concrete lexical entry in (2), 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.

Definition 10 (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 (2), 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.

Definition 11 (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 $(dl\ d_1) \to 2^{(dl\ d_2)}$ mapping labels on d_1 to sets of labels on d_2.*

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

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 (2), $d_1 = \text{SEM}$ 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.

3.4 Models

Definition 12 (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 13 (Lexicon Satisfaction) *Given a grammar $G = (MT, lex, P)$, an XDG 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:*

$$
\begin{aligned}
&\forall v \in V : \exists e \in lex\ (nw\ v) : \forall d \in D : \forall a \in A' : \\
&(na\ v\ d\ a) = (e\ d\ a)
\end{aligned}
$$

Definition 14 (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.*

3.5 String Language

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

Definition 15 (Yield of a Multigraph) *The yield of $M = (V, E^+, <, nw, na)$ is the concatenation of the words of the nodes, ordered with respect to the strict total order $<$:*

$$y\,M \quad = \quad nw\,p_i \;\ldots\; nw\,p_{|V|}$$

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

Definition 16 (String Language) *The string language $L\,G$ of a grammar G is the set of yields of the models of G:*

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

Although parsing is not the topic of this paper, this 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, but crucially, no nodes are added.

4 Recognition Problems and their Complexity

Definition 17 (XDG Recognition Problem (RP)) *Given a grammar G and a string s, is s in $L(G)$?*

We distinguish the following three flavors of the RP, including the new *instance recognition problem* for instances of XDG:

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

Definition 18 (Generate and Test Recognition) *A simple recognition algorithm for XDG is* generate and test*: given a grammar $G = (MT, lex, P)$ and a string s, we 1) non-deterministically guess a multigraph for s of multigraph type MT, and 2) verify whether the multigraph satisfies the lexicon*[1] *and the principles. Since both are in FO, verifying amounts to FO* model checking.

For model checking, Vardi (1982) distinguishes three complexity measures: *data complexity* (defined with respect to the model size), *expression complexity* (formula size), and *combined expressivity* (model and formula size). For FOL, data complexity is in LOGSPACE, and expression and combined complexity in PSPACE. For FOL without quantifiers, all three measures are in LOGSPACE.

We use Vardi's results to establish upper bounds of the complexity of the verification step of Definition 18 for all three flavors of RPs:

[1] We only need to consider the part of the lexicon for the words in s.

1. URP: G and s are variable. Variable G corresponds to variable formula size of the lexicon *lex* and the principles in P, and variable s to variable model size of the guessed multigraph for s. The relevant measure is FOL combined complexity.
2. FRP: s is variable, corresponding to variable model size of the guessed multigraph for s. Relevant measure: FOL data complexity.
3. IRP: *lex* and s are variable. Variable *lex* corresponds to variable formula size of the lexicon. Testing for lexicon satisfaction can obviously be done in linear time. Variable s corresponds to variable model size of the guessed multigraph for s, measurable by FOL data complexity.

We now turn to establishing the complexity of the three flavors of RPs. First, we prove that the URP is PSPACE-hard by a reduction of the QSAT problem.

Definition 19 (Quantified Satisfiability Problem (QSAT)) *An instance of QSAT is a quantified Boolean formula with no free variables:*

$$Q_1 X_1 \ldots Q_n X_n F$$

where Q stands for either $\exists$ or $\forall$, and F for a boolean formula. The QSAT problem is to decide whether the quantified formula is true.

Lemma 1 (The URP is PSPACE-hard) *For this lemma, we construct a polynomial-time reduction from instances of QSAT to instances of the URP. For each new string s, the URP allows us to construct a new grammar G, and in particular, to construct new principles. As QSAT formulas f can be rephrased in FOL with equality, we can for each rephrased QSAT formula f' construct a grammar $G = (MT, lex, \{f'\})$, where the rephrased QSAT formula is the only principle. The question whether any QSAT formula f is true can then be reformulated as the URP for the constructed grammar G and an arbitrarily chosen string s.*

Lemma 2 (The URP is in PSPACE) *The relevant measure for the complexity of the verification step of the URP is FOL combined complexity, which is in PSPACE. The combination of non-deterministically guessing a multigraph and verifying it is hence in NPSPACE, which is =PSPACE by Savitch's Theorem.*

Theorem 1 (The URP is PSPACE-complete) *Follows from lemma 1 and Lemma 2.*

Lemma 3 (The FRP is NP-hard) *This is proven in (Debusmann 2006) using a reduction of the Satisfiability problem (SAT) to a fixed XDG grammar. We can rephrase the grammar in FO XDG (not shown here for lack of space), and thus re-use the result.*

Lemma 4 (The FRP is in NP) *The relevant measure for the complexity of the verification step of the FRP is FOL data complexity, which is in LOGSPACE, i.e., in P. The combination of non-deterministically guessing a multigraph and verifying it is hence in NP.*

Theorem 2 (The FRP is NP-complete) *Follows from Lemma 3 and Lemma 4.*

Lemma 5 (The IRP is NP-hard) *Follows from Lemma 3.*

Lemma 6 (The IRP is in NP) *The IRP verification step can be done in polynomial time: 1) testing for lexicon satisfaction can be done in linear time, and 2) FOL data complexity is in LOGSPACE. The combination of non-deterministically guessing a multigraph and verifying it is hence in NP.*

Theorem 3 (The IRP is NP-complete) *Follows from Lemma 5 and Lemma 6.*

5 Conclusions and Related Work

Taking advantage of a new formalization of XDG in FOL, we have established lower and upper bounds for three flavors of the XDG RP. The results are useful for comparing XDG with other grammar formalisms, where mostly their URP is considered. The URP for other grammar formalisms is however not directly comparable with the URP for XDG, which is about XDG as a *meta* grammar formalism. The better choice is the IRP for *instances* of it. NP-completeness of the IRP places XDG right between a number of more complex and a number of less complex grammar formalisms. For unification-based grammars such as HPSG, the URP is generally undecidable (Trautwein 1995). The URP for mildly context-sensitive grammar formalisms such as CCG and TAG on the other hand is in P. Interestingly, even for the "intractable" grammar formalisms such as HPSG and LFG there exist parsers which are efficient in practice. Similarly, the XDG constraint parser is efficient for smaller-scale handcrafted grammars and the large-scale XTAG grammar (Debusmann, forthcoming).

The new complexity results also shed light on the *expressivity* of XDG. As for context-sensitive grammars (CSG), the URP for XDG is PSPACE-complete. However, for CSG, the FRP is also PSPACE-complete, whereas the FRP for XDG is NP-complete. This suggests that context-sensitive languages provide a strict upper bound for the expressivity of XDG.

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.

References

Debusmann, R. (2006), Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description, PhD thesis, Universität des Saarlandes.

Debusmann, R., Duchier, D., Koller, A., Kuhlmann, M., Smolka, G. & Thater, S. (2004), A relational syntax-semantics interface based on dependency grammar, *in* 'Proceedings of COLING 2004', Geneva/CH.

Debusmann, R., Postolache, O. & Traat, M. (2005), A modular account of information structure in Extensible Dependency Grammar, *in* 'Proceedings of the CICLING 2005 Conference', Springer, Mexico City/MX.

Duchier, D. & Debusmann, R. (2001), Topological dependency trees: A constraint-based account of linear precedence, *in* 'Proceedings of ACL 2001', Toulouse/FR.

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

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

Jackendoff, R. (2002), *Foundations of Language*, Oxford University Press.

Koller, A. & Striegnitz, K. (2002), Generation as dependency parsing, *in* 'Proceedings of ACL 2002', Philadelphia/US.

Lison, P. (2006), Implémentation d'une interface sémantique-syntaxe basée sur des grammaires d'unification polarisées, Master's thesis, Univesité Catholique de Louvain.

Rogers, J. (1996), A model-theoretic framework for theories of syntax, *in* 'Proceedings of ACL 1996'.

Steedman, M. (2000), 'Information structure and the syntax-phonology interface', *Linguistic Inquiry* **31**(4), 649–689.

Tesnière, L. (1959), *Eléments de Syntaxe Structurale*, Klincksiek, Paris/FR.

Trautwein, M. (1995), The complexity of structure sharing in unification-based Grammars, *in* W. Daelemans, G. Durieux & S. Gillis, eds, 'Computational Linguistics in the Netherlands 1995', pp. 165–179.

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