Multi-dimensional Dependency Grammar as Multigraph Description

Ralph Debusmann and **Gert Smolka**
Programming Systems Lab
Universität des Saarlandes
Postfach 15 11 50
66041 Saarbrücken, Germany
{rade,smolka}@ps.uni-sb.de

Abstract

Extensible Dependency Grammar (*XDG*) is new, modular grammar formalism for natural language. An XDG analysis is a *multi-dimensional dependency graph*, where each dimension represents a different aspect of natural language, e.g. *syntactic function*, *predicate-argument structure*, *information structure* etc. Thus, XDG brings together two recent trends in computational linguistics: the increased application of ideas from *dependency grammar* and the idea of *multi-layered linguistic description*. In this paper, we tackle one of the stumbling blocks of XDG so far—its incomplete formalization. We present the first complete formalization of XDG, as a description language for *multigraphs* based on *simply typed lambda calculus*.

Introduction

Extensible Dependency Grammar (*XDG*) (Debusmann *et al.* 2004) brings together two recent trends from computational linguistics:

1. *dependency grammar*
2. *multi-layered linguistic description*

Firstly, the ideas of *dependency grammar*, *lexicalization*, the *head-dependent asymmetry*, *valency* etc., have become more and more popular in computational linguistics. Most of the popular grammar formalisms like *Combinatorial Categorial Grammar* (*CCG*) (Steedman 2000), *Head-driven Phrase Structure Grammar* (*HPSG*) (Pollard & Sag 1994), *Lexical Functional Grammar* (*LFG*) (Bresnan 2001) and *Tree Adjoining Grammar* (*TAG*) (Joshi 1987) have already adopted these ideas. Moreover, the most successful approaches statistical parsing crucially depend on notions from dependency grammar (Collins 1999), and new treebanks based on dependency grammar are being developed for various languages, e.g. the *Prague Dependency Treebank* (*PDT*) for Czech and the *TiGer Dependency Bank* for German.

Secondly, many treebanks such as the *Penn Treebank*, the *TiGer Treebank* and the PDT are continuously being extended with additional layers of annotation in addition to the syntactic layer, i.e. they become more and more *multi-layered*. For example, the *PropBank* (Kingsbury & Palmer 2002) (Penn Treebank), the *SALSA project* (Erk *et al.* 2003) (TiGer Treebank) and the *tectogrammatical layer* (PDT) add a layer of predicate-argument structure. Other added layers concern *information structure* (PDT) and *discourse structure* as in the *Penn Discourse Treebank* (Webber *et al.* 2005). These additional layers of annotation are often dependency-like, i.e. could be straightforwardly represented in a framework for dependency grammar which is multi-layered.

XDG is such a framework. It has already been successfully applied to model a relational syntax-semantics interface (Debusmann *et al.* 2004) and to model the relation between *prosodic structure* and *information structure* in English (Debusmann, Postolache, & Traat 2005). We hope to soon be able to employ XDG to directly make use of the information contained in the new multi-layered treebanks, e.g. for the automatic induction of multi-layered grammars for parsing and generation.

To achieve this goal, XDG still needs to overcome a number of weaknesses. The first is the lack of a *polynomial parsing algorithm*—so far, we only have a parser based on *constraint programming* (Debusmann, Duchier, & Niehren 2004), which is fairly efficient, given that the parsing problem is *NP-hard*, but does not scale up to large-scale grammars. The second major stumbling block of XDG so far is the lack of a complete formalization. The latter is what we will change in this paper: we will present a formalization of XDG as a description language for *multigraphs* based on *simply typed lambda calculus* (Church 1940; Andrews 2002). To give a hint of the expressivity of XDG, we additionally present a proof that the parsing problem of (unrestricted) XDG is *NP-hard*. We begin the paper with introducing the notion of *multigraphs*.

Multigraphs

Multigraphs are motivated by *dependency grammar*, and in particular by its structures: *dependency graphs*.

Dependency Graphs

Dependency graphs such as the one in Figure 1 typically represent the syntactic structure of sentences in natural language. They have the following properties:

1. Each *node* (round circle) is associated with a *word* (*today*, *Peter*, *wants* etc.), which is connected to the corresponding node by a dotted vertical line called *projection edge*,

for these lines signify the projection of the nodes onto the sentence.

2. Each node is additionally associated with an *index* (1, 2, 3 etc.) signifying its position in the sentence, and displayed above the words.
3. The nodes are connected to each other by labeled and directed *edges*, which express syntactic relations. In the example, there is an edge from node 3 to node 1 labeled adv, expressing that *wants* is modified by the adverb *today*, and one from 3 to 2 labeled subj, expressing that *Peter* is the subject of *wants*. *wants* also has the infinitival verbal complement (edge label vinf) *eat*, which has the particle (part) *to* and the object (obj) *spaghetti*.

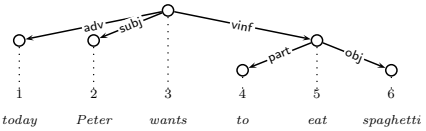


Figure 1: Dependency Graph (syntactic analysis)

Dependency graphs are not restricted to describing syntactic structures alone. As an example in point, Figure 2 shows a dependency graph which describes the *semantic structure* of our example sentence, where the adverb *today* is the root and has the *theme* (edge label th) *wants*, which in turn has the *agent* (ag) *Peter* and the theme *eat*. *eat* has the agent *Peter* and the *patient* (pat) *spaghetti*. Node 4 (the particle *to*) has no meaning and thus remains unconnected.

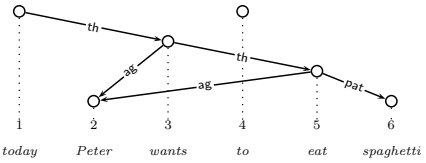


Figure 2: Dependency Graph (semantic analysis)

Multigraphs

A *multigraph* is a *multi-dimensional* dependency graph consisting of an arbitrary number of dependency graphs called *dimensions*. All dimensions share the same set of nodes. Multigraphs can significantly simplify linguistic modeling by modularizing both the structures and their description.

We show an example multigraph in Figure 3, where we simply draw the dimensions of *syntax* (upper half, from Figure 1) and *semantics* (lower half, from Figure 2) as individual dependency graphs for clarity, and indicate the node sharing by arranging shared nodes in the same columns. The multigraph simultaneously describes the syntactic and semantic structures of the sentence, and expresses e.g. that *Peter* (node 2), the subject of *wants*, syntactically realizes both the agent of *wants* and of *eat* (node 5).

Formalization

We formalize multigraphs as tuples $(V, Dim, Word, W, Lab, E, Attr, A)$ of a finite set V

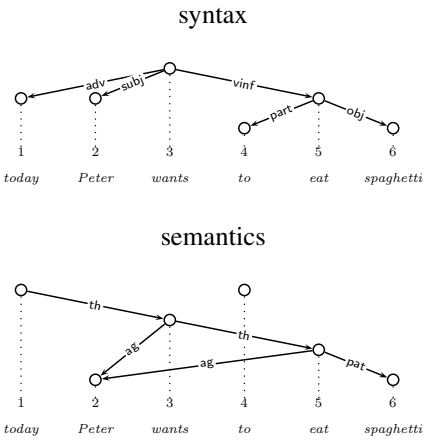


Figure 3: Multigraph

of *nodes*, a finite set Dim of *dimensions*[1], a finite set $Word$ of *words*, the *node-word mapping* $W \in V \rightarrow Word$, a finite set Lab of *edge labels*, a set $E \subseteq V \times V \times Dim \times Lab$ of *edges*, and in addition a finite set $Attr$ of *attributes* and the *node-attributes mapping* $A \in V \rightarrow Dim \rightarrow Attr$ to equip the nodes with extra information (not used in the example above). The set of nodes V must be a finite interval of the natural numbers starting from 1, i.e., given n nodes, $V = \{1, \ldots, n\}$, which induces a strict total order on V.

Relations

We associate each dimension $d \in Dim$ with four relations: 1) the *labeled edge relation* ($\dot{\longrightarrow}_d$), 2) the *edge relation* ($\rightarrow_d$), 3) the *dominance relation* ($\rightarrow_d^+$), and 4) the *precedence relation* ($\prec$).

Labeled Edge. Given two nodes v and v' and a label $l \in Lab$, the *labeled edge relation* $v \xrightarrow{l}_d v'$ holds iff there is an edge from v to v' labeled l on dimension d:

$$\dot{\longrightarrow}_d \quad = \quad \{(v, v', l) \mid (v, v', d, l) \in E\} \qquad (1)$$

Edge. Given two nodes v and v', the *edge relation* $v \rightarrow_d v'$ holds iff there is an edge from v to v' on dimension d labeled by any label in Lab:

$$\rightarrow_d \quad = \quad \{(v, v') \mid \exists l \in Lab : v \xrightarrow{l}_d v'\} \qquad (2)$$

Dominance. The (strict, non-reflexive) *dominance relation* $\rightarrow_d^+$ is the transitive closure of the edge relation $\rightarrow_d$.

Precedence. Given two nodes v and v', the *precedence relation* $v \prec v'$ holds iff $v < v'$, where $<$ is the total order on the natural numbers.

[1]Note that here, formally, a dimension is just a name identifying a particular dependency graph in the multigraph. Conceptually, a dimension denotes not just the name but the dependency graph itself.

A Description Language for Multigraphs

We proceed with formalizing *Extensible Dependency Grammar* (*XDG*) as a description language for multigraphs based based on *simply typed lambda calculus*.

Types

Definition. We define the types $T \in Ty$ given a set At of *atoms* (arbitrary symbols):

$$\begin{array}{lrll} & a \in At & & \\ T \in Ty & ::= & \mathsf{B} & \textit{boolean} \\ & | & \mathsf{V} & \textit{node} \\ & | & T_1 \to T_2 & \textit{function} \\ & | & \{a_1, \ldots, a_n\} & \textit{finite domain} \\ & | & \{a_1 : T_1, \ldots, a_n : T_n\} & \textit{record} \end{array} \tag{3}$$

where $n \geq 1$ and for finite domains and records, $a_1, \ldots, a_n$ are pairwise distinct. Following (Church 1940; Andrews 2002), we adopt the classical semantics for lambda calculus and thus forbid empty finite domain types.

Interpretation. We interpret:

- B as $\{0, 1\}$
- V as a finite interval of the natural numbers from 1
- $T_1 \to T_2$ as the set of all functions from the interpretation of T_1 to the interpretation of T_2
- $\{a_1, \ldots, a_n\}$ as the set $\{a_1, \ldots, a_n\}$
- $\{a_1 : T_1, \ldots, a_n : T_n\}$ as the set of all functions f with
  1. $Dom\ f = \{a_1, \ldots, a_n\}$
  2. for all $1 \leq i \leq n$, $f\ a_i$ is an element of the interpretation of T_i

Multigraph Type

Multigraphs can be distinguished according to their dimensions, words, edge labels and attributes. This leads us to the definition of a *multigraph type*, which we define given the types Ty as a tuple $M = (dim, word, lab, attr)$, where

1. $dim \in Ty$ is a finite domain of *dimensions*
2. $word \in Ty$ is a finite domain of *words*
3. $lab \in dim \to Ty$ is a function from dimensions to *label types* (finite domains), i.e. the type of the edge labels on that dimension
4. $attr \in dim \to Ty$ is a function from dimensions to *attributes types* (any type in Ty), i.e. the type of the attributes on that dimension

Writing $\mathcal{M}\ T$ for the interpretation of type T over $\mathcal{M}$, a multigraph $\mathcal{M} = (V, Dim, Word, W, Lab, E, Attr, A)$ has multigraph type $M = (dim, word, lab, attr)$ iff

1. The dimensions are the same:
$$Dim \quad = \quad \mathcal{M}\ dim \tag{4}$$
2. The words are the same:
$$Word \quad = \quad \mathcal{M}\ word \tag{5}$$
3. The edges in E have the right edge labels for their dimension (according to lab):
$$\forall (v, v', d, l) \in E \quad : \quad l \in \mathcal{M}\ (lab\ d) \tag{6}$$
4. The nodes have the right attributes for their dimension (according to $attr$):
$$\forall v \in V \quad : \quad \forall d \in Dim \quad : \quad (A\ v\ d) \in \mathcal{M}\ (attr\ d) \tag{7}$$

Terms

The *terms* of XDG augment simply typed lambda calculus with *atoms*, *records* and *record selection*. Given a set of constants Con, we define:

$$\begin{array}{lrll} & a \in At & & \\ & c \in Con & & \\ t & ::= & x & \text{variable} \\ & | & c & \text{constant} \\ & | & \lambda x : T.t & \text{abstraction} \\ & | & t_1\ t_2 & \text{application} \\ & | & a & \text{atom} \\ & | & \{a_1 = t_1, \ldots, a_n = t_n\} & \text{record} \\ & | & t.a & \text{record selection} \end{array} \tag{8}$$

where for records, $a_1, \ldots, a_n$ are pairwise distinct.

Signature

An XDG signature is determined by a *multigraph type* $M = (dim, word, lab, attr)$, and consists of two parts: the *logical constants* and the *multigraph constants*.

Logical Constants. The logical constants include the type constant B and the following term constants:

$$\begin{array}{rcll} 0 & : & \mathsf{B} & \textit{false} \\ 1 & : & \mathsf{B} & \textit{true} \\ \neg & : & \mathsf{B} \to \mathsf{B} & \textit{negation} \\ \vee & : & \mathsf{B} \to \mathsf{B} \to \mathsf{B} & \textit{disjunction} \\ \doteq_T & : & T \to T \to \mathsf{B} & \textit{equality} \\ \exists_T & : & (T \to \mathsf{B}) \to \mathsf{B} & \textit{existential quantification} \end{array} \tag{9}$$

For convenience, we introduce the usual *logical constants* $\wedge, \Rightarrow, \Leftrightarrow, \neq, \exists^1_T$ and $\forall_T$ as notation.

Multigraph Constants. The multigraph constants include the type constant V, and the following term constants:

$$\begin{array}{rcll} \dot{\longrightarrow}_d & : & \mathsf{V} \to \mathsf{V} \to lab\ d \to \mathsf{B} & \textit{labeled edge} \\ \to_d & : & \mathsf{V} \to \mathsf{V} \to \mathsf{B} & \textit{edge} \\ \to^+_d & : & \mathsf{V} \to \mathsf{V} \to \mathsf{B} & \textit{dominance} \\ \prec & : & \mathsf{V} \to \mathsf{V} \to \mathsf{B} & \textit{precedence} \\ (word\ \cdot) & : & \mathsf{V} \to word & \textit{word} \\ (d\ \cdot) & : & \mathsf{V} \to attr\ d & \textit{attributes} \end{array} \tag{10}$$

where we interpret

- $\dot{\longrightarrow}_d$ as the *labeled edge relation* on dimension d.
- $\to_d$ as the *edge relation* on d.

- $\rightarrow_d^+$ as the *dominance relation* on d.
- $\prec$ as the *precedence relation*
- $(word\ \cdot)$ as the *word*
- $(d\ \cdot)$ as the *attributes* on d.

Grammar

An *XDG grammar* $G = (M, P)$ is defined by a *multigraph type* M and a set P of *formulas* called *principles*. Each principle must be formulated according to the signature M.

Models

The *models* of a grammar $G = (M, P)$ are all multigraphs which

1. have multigraph type M
2. satisfy all principles P

Recognition Problem

We distinguish two kinds of *recognition problems*:

1. the *universal recognition problem*
2. the *fixed recognition problem*

Universal Recognition Problem. Given an XDG grammar G with words $word$ and a string $s = w_1 \dots w_n$ in $word^+$, the *universal recognition problem* (G, s) is the problem of determining whether there is a model of G such that:

1. there are as many nodes as words:
$$V \quad = \quad \{1, \dots, n\} \tag{11}$$
2. the concatenation of the words of the nodes yields s:
$$(word\ 1) \dots (word\ n) \quad = \quad s \tag{12}$$

Fixed Recognition Problem. The *fixed recognition problem* (G, s) poses the same question as the universal recognition problem, but with the restriction that for all input strings s, the grammar G must remain fixed.

Complexity

What is the *complexity* of the two kinds of recognition problems? In this section, we prove that the fixed recognition problem is *NP-hard*. The purpose of the proof is to give the reader a feeling for the expressivity of XDG.

Fixed Recognition Problem

Proof. To prove that the fixed string membership problem is *NP-hard*, we opt for the reduction of the NP-complete problem of SAT (satisfiability), which is the problem of deciding whether a formula in propositional logic has an assignment that evaluates to true. We restrict ourselves to formulas f, which is already sufficient to cover full propositional logic:

$$\begin{array}{rll} f ::= & X, Y, Z, \dots & \textit{variable} \\ \mid & 0 & \textit{false} \\ \mid & f_1 \Rightarrow f_2 & \textit{implication} \end{array} \tag{13}$$

The reduction of SAT proceeds as follows:

1. In three steps, we transform the propositional formula into a string suitable as input to the fixed string membership problem. For example, given the formula
$$(X \Rightarrow 0) \Rightarrow Y \tag{14}$$
the transformation goes along as follows:
   (a) To avoid ambiguity, we transform the formula into *prefix notation*:
   $$\Rightarrow \Rightarrow\ X\ 0\ Y \tag{15}$$
   (b) A propositional formula can contain an arbitrary number of variables, yet the domain of words of an XDG grammar must be finite. To overcome this limitation, we adopt a unary encoding for variables, where we encode the first variable from the left of the formula (here: X) as *var* I, the second (here: Y) *var* I I etc.:
   $$\Rightarrow \Rightarrow\ \textit{var}\ I\ 0\ \textit{var}\ I\ I \tag{16}$$
   (c) To clearly distinguish the string from the original propositional formula, we replace all implication symbols with the word *impl*:
   $$\textit{impl impl var}\ I\ 0\ \textit{var}\ I\ I \tag{17}$$
2. We introduce the *Propositional Logic dimension* (abbreviation: PL) to model the structure of the propositional formula. On the PL dimension, the example formula (14) is analyzed as in Figure 4.

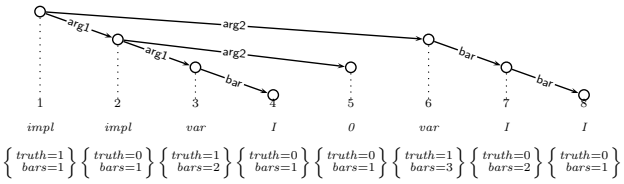


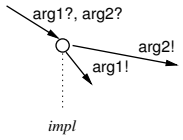
Figure 4: PL analysis of $(X \Rightarrow 0) \Rightarrow Y$

The edge labels are arg1 and arg2 for the *antecedent* and the *consequent* of an implication, respectively, and bar for connecting the bars (word I) of the unary variable encoding. Below the words of the nodes, we display their attributes, which are of the following type:

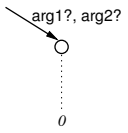
$$\left\{ \begin{array}{rl} truth & : \mathsf{B} \\ bars & : \mathsf{V} \end{array} \right\} \tag{18}$$

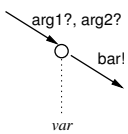
where $truth$ represents the truth value of the node and $bars$ the number of bars below the next node to its right plus 1. The $bars$ attribute will become crucial for establishing coreferences between variables. Its type is V for two reasons:

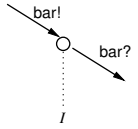
(a) There are always less (or equally many) variables in a formula than there are nodes, since every encoded formula contains less (or equally many) variables than words. Hence, V always suffices to distinguish them.
(b) We require the *precedence predicate* to implement *incrementation* (cf. 9. below), which is defined only on V.

3. We require that the models on PL are *trees*. In XDG, we can express this as follows:
   (a) there are no cycles:
   $$\neg(v \rightarrow^{+}_{\text{PL}} v) \tag{19}$$
   (b) each node has at most one incoming edge:
   $$v' \rightarrow_{\text{PL}} v \;\wedge\; v'' \rightarrow_{\text{PL}} v \;\Rightarrow\; v' \doteq v'' \tag{20}$$
   (c) there is precisely one node with no incoming edge (the root):
   $$\exists^1 v : \neg\exists v' : v' \rightarrow_{\text{PL}} v \tag{21}$$
4. We describe the structure of the PL tree by, for each node, depending on its word, constraining 1) its incoming edges, 2) its outgoing edges and 3) its order with respect to its daughters and the order among the daughters:
   (a) A node with word *impl* 1) can either be the antecedent or the consequent of an implication (its incoming edge must be labeled either arg1 or arg2), 2) requires precisely one outgoing edge labeled arg1 (for the antecedent) and one labeled arg2 (for the consequent) and must have no other outgoing edges, and 3) must precede its arg1-daughter, which in turn must precede the arg2-daughter. We illustrate this below in (22), where the question mark ? marks optional and the exclamation mark ! obligatory edges:
   

   (22)

   We can express these three constraints in XDG as:
   $$\begin{aligned}&(\mathit{word}\ v) \doteq \mathit{impl} \Rightarrow\\ &1)\quad v' \xrightarrow{l}_{\text{PL}} v \Rightarrow l \doteq \mathsf{arg1} \vee l \doteq \mathsf{arg2} \quad \wedge\\ &2)\quad \exists^1 v' : v \xrightarrow{\mathsf{arg1}}_{\text{PL}} v' \wedge \exists^1 v' : v \xrightarrow{\mathsf{arg2}}_{\text{PL}} v' \quad \wedge\\ &\quad\ \neg\exists v' : v \xrightarrow{\mathsf{bar}}_{\text{PL}} v' \quad \wedge\\ &3)\quad v \xrightarrow{\mathsf{arg1}}_{\text{PL}} v' \wedge v \xrightarrow{\mathsf{arg2}}_{\text{PL}} v'' \Rightarrow v \prec v' \wedge v' \prec v''\end{aligned} \tag{23}$$
   (b) A node with word 0 can 1) either be the antecedent or the consequent of an implication and 2) must not have any outgoing edges:
   

   (24)
   (c) A node with word *var* 1) can either be the antecedent or the consequent of an implication, 2) requires only precisely one outgoing edge labeled bar for the first bar below it, and 3) precedes its bar-daughter:
   

   (25)
   (d) A node with word I 1) must have an incoming edge labeled bar, 2) can have only at most one outgoing edge labeled bar, and 3) precedes its potential bar-daughter:
   

   (26)
5. Just ordering the nodes is not enough: to precisely capture the context-free structure of the propositional formula and the unary variable encoding, we must ensure that the models are *projective*, i.e. that no edge crosses any of the projection edges.[2] Otherwise, we are faced with wrong analyses as in Figure 5, where the rightmost bar (node 8) is incorrectly "taken over" by the first bar (node 4) of the left variable (node 3).

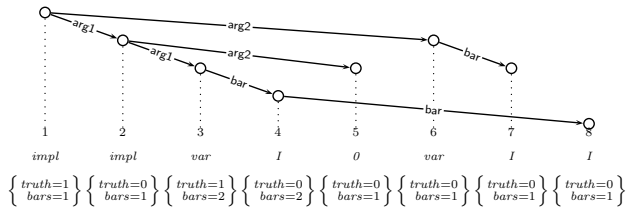
   

   Figure 5: Non-projective PL analysis of $(X \Rightarrow 0) \Rightarrow Y$

   We express the projectivity constraint as follows in XDG: for each node v and each daughter v', all nodes v'' between v and v' must be below v:
   $$\begin{aligned}&v \rightarrow_{\text{PL}} v' \wedge v \prec v' \quad\Rightarrow\quad \forall v'' : v \prec v'' \wedge v'' \prec v' \quad\Rightarrow\\ &\qquad\qquad\qquad\qquad v \rightarrow^{+}_{\text{PL}} v'' \quad\wedge\\ &v \rightarrow_{\text{PL}} v' \wedge v' \prec v \quad\Rightarrow\quad \forall v'' : v' \prec v'' \wedge v'' \prec v \quad\Rightarrow\\ &\qquad\qquad\qquad\qquad v \rightarrow^{+}_{\text{PL}} v''\end{aligned} \tag{27}$$
6. We must ensure that the root of PL analysis always has truth value 1, i.e. that the propositional formula is true:
   $$\neg\exists v' : v' \rightarrow_{\text{PL}} v \Rightarrow (\text{PL}\ v).\mathit{truth} \doteq 1 \tag{28}$$
7. Nodes with word 0 have truth value false. Their bar count is not relevant, hence we can pick an arbitrary value and set it to 1:
   $$\begin{aligned}&(\mathit{word}\ v) \doteq 0 \Rightarrow\\ &(\text{PL}\ v).\mathit{truth} \doteq 0 \quad\wedge\\ &(\text{PL}\ v).\mathit{bars} \doteq 1\end{aligned} \tag{29}$$
8. The truth value of nodes with word *impl* equals the implication of the truth value of its arg1-daughter (the antecedent) and its arg2-daughter (the consequent). Again the bar count is not relevant and we set it to 1:
   $$\begin{aligned}&(\mathit{word}\ v) \doteq \mathit{impl} \Rightarrow\\ &(v \xrightarrow{\mathsf{arg1}}_{\text{PL}} v' \wedge v \xrightarrow{\mathsf{arg2}}_{\text{PL}} v'' \Rightarrow\\ &(\text{PL}\ v).\mathit{truth} \doteq ((\text{PL}\ v').\mathit{truth} \Rightarrow (\text{PL}\ v'').\mathit{truth})) \quad\wedge\\ &(\text{PL}\ v).\mathit{bars} \doteq 1\end{aligned} \tag{30}$$

[2] Note that while XDG dimensions *can* be constrained to be projective, they need not be.

9. The truth value of bars (word I) is not relevant, and hence we can safely set it to 0. The bar value is either 1 for the bars not having a daughter, or else one more than its daughter:

$$\begin{array}{l}(word\ v) \doteq I \Rightarrow \\ (\text{PL}\ v).truth \doteq 0 \quad \wedge \\ \neg\exists v' : v \rightarrow_{\text{PL}} v' \Rightarrow (\text{PL}\ v).bars \doteq 1 \quad \wedge \\ (v \xrightarrow{\mathsf{bar}}_{\text{PL}} v' \Rightarrow (\text{PL}\ v').bars \prec (\text{PL}\ v).bars \wedge \\ \neg\exists v'' : (\text{PL}\ v').bars \prec v'' \wedge v'' \prec (\text{PL}\ v).bars)\end{array} \tag{31}$$

Notice that the latter constraint actually *increments* the bar value, even though XDG does not provide us with any direct means to do that. The trick is to emulate incrementing using the precedence predicate.

10. The truth value of variables is only constrained by the overall propositional formula. Their bar value is the same as that of their bar daughter.

$$\begin{array}{l}(word\ v) \doteq var \Rightarrow \\ v \xrightarrow{\mathsf{bar}}_{\text{PL}} v' \Rightarrow (\text{PL}\ v).bars \doteq (\text{PL}\ v').bars\end{array} \tag{32}$$

11. We can now establish coreferences between the variables. To this end, we stipulate that for each pair of nodes v and v' with word *var*, if they have the same bar values, then their truth values must be the same:

$$\begin{array}{l}(word\ v) \doteq var \wedge (word\ v') \doteq var \Rightarrow \\ (\text{PL}\ v).bars \doteq (\text{PL}\ v').bars \Rightarrow \\ (\text{PL}\ v).truth \doteq (\text{PL}\ v').truth\end{array} \tag{33}$$

The described reduction is polynomial, which concludes the proof that the fixed string membership problem is NP-hard. □

Universal Recognition Problem

Proof. Since the fixed recognition problem is a specialization of the universal recognition problem, the NP-hardness result carries over: the universal recognition problem of XDG is also NP-hard. □

Conclusion

In this paper, we have presented the first complete formalization of XDG as a description language for multigraphs based on simply typed lambda calculus. We also showed that the recognition problem of XDG as it stands is NP-hard. But all is not lost: on the one hand, we already have an implementation of XDG as a constraint satisfaction problem which is fairly efficient for handcrafted grammars at least. Also, other grammar formalisms such as LFG (Barton, Berwick, & Ristad 1987) and HPSG (with significant restrictions) (Trautwein 1995) are also NP-hard but still used for parsing in practice. On the other hand, the formalization is meant to be a starting point for future research on the formal aspects of XDG, with the eventual goal to find more tractable, polynomial fragments, without losing the potential to significantly modularize and thus improve the modeling of natural language.

Acknowledgments

We would like to thank the other members of the CHORUS project (Alexander Koller, Marco Kuhlmann, Manfred Pinkal, Stefan Thater) for discussion, and also Denys Duchier and Joachim Niehren. This work was funded by the DFG (SFB 378) and the International Post-Graduate College in Language Technology and Cognitive Systems (IGK) in Saarbrücken.

References

Andrews, P. B. 2002. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers.

Barton, G. E.; Berwick, R.; and Ristad, E. S. 1987. *Computational Complexity and Natural Language*. MIT Press.

Bresnan, J. 2001. *Lexical Functional Syntax*. Blackwell.

Church, A. 1940. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic* (5):56–68.

Collins, M. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. Dissertation, University of Pennsylvania.

Debusmann, R.; Duchier, D.; Koller, A.; Kuhlmann, M.; Smolka, G.; and Thater, S. 2004. A Relational Syntax-Semantics Interface Based on Dependency Grammar. In *Proceedings of COLING 2004*.

Debusmann, R.; Duchier, D.; and Niehren, J. 2004. The XDG Grammar Development Kit. In *Proceedings of the MOZ04 Conference*, volume 3389 of *Lecture Notes in Computer Science*, 190–201. Charleroi/BE: Springer.

Debusmann, R.; Postolache, O.; and Traat, M. 2005. A Modular Account of Information Structure in Extensible Dependency Grammar. In *Proceedings of the CICLING 2005 Conference*. Mexico City/MX: Springer.

Erk, K.; Kowalski, A.; Pado, S.; and Pinkal, M. 2003. Towards a Resource for Lexical Semantics: A Large German Corpus with Extensive Semantic Annotation. In *Proceedings of ACL 2003*.

Joshi, A. K. 1987. An Introduction to Tree-Adjoining Grammars. In Manaster-Ramer, A., ed., *Mathematics of Language*. Amsterdam/NL: John Benjamins. 87–115.

Kingsbury, P., and Palmer, M. 2002. From Treebank to PropBank. In *Proceedings of LREC-2002*.

Pollard, C., and Sag, I. A. 1994. *Head-Driven Phrase Structure Grammar*. Chicago/US: University of Chicago Press.

Steedman, M. 2000. *The Syntactic Process*. Cambridge/US: MIT Press.

Trautwein, M. 1995. The complexity of structure sharing in unification-based Grammars. In Daelemans, W.; Durieux, G.; and Gillis, S., eds., *Computational Linguistics in the Netherlands 1995*, 165–179.

Webber, B.; Joshi, A.; Miltsakaki, E.; Prasad, R.; Dinesh, N.; Lee, A.; and Forbes, K. 2005. A Short Introduction to the Penn Discourse TreeBank. Technical report, University of Pennsylvania.