Second ACL Workshop on Multiword Expressions: Integrating Processing, July 2004, pp. 56-63

Multiword expressions as dependency subgraphs

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

Abstract

We propose to model multiword expressions as dependency subgraphs, and realize this idea in the grammar formalism of Extensible Dependency Grammar (XDG). We extend XDG to lexicalize dependency subgraphs, and show how to compile them into simple lexical entries, amenable to parsing and generation with the existing XDG constraint solver.

1 Introduction

In recent years, dependency grammar (DG) (Tesnière, 1959; Sgall et al., 1986; Mel'čuk, 1988) has received resurgent interest. Core concepts such as grammatical functions, valency and the head-dependent asymmetry have now found their way into most grammar formalisms, including phrase structure-based ones such as HPSG, LFG and TAG. This renewed interest in DG has also given rise to new grammar formalisms based directly on DG (Nasr, 1995; Heinecke et al., 1998; Bröker, 1999; Gerdes and Kahane, 2001; Kruijff, 2001; Joshi and Rambow, 2003).

A controversy among DG grammarians circles around the question of assuming a 1:1-correspondence between words and nodes in the dependency graph. This assumption simplifies the formalization of DGs substantially, and is often required for parsing. But as soon as semantics comes in, the picture changes. Clearly, the 1:1-correspondence between words and nodes does not hold anymore for multiword expressions (MWEs), where one semantic unit, represented by a node in a semantically oriented dependency graph, corresponds not to one, but to more than one word.

Most DGs interested in semantics have thus weakened the 1:1-assumption, starting with Tesnière's work. Tesnière proposed the concept of *nuclei* to group together sets of nodes. FGD, on the other hand, allows for the deletion of solely syntactically motivated nodes in the *tectogrammatical structure*. Similarly, in MTT, nodes present on the syntactic structures can be deleted on the semantic structure. This can happen e.g. through paraphrasing rules implemented by *lexical functions* (Mel'čuk, 1996). Unfortunately, these attempts to break the 1:1-correspondence have not yet been formalized in a declarative way.

Extensible Dependency Grammar (XDG) is a new grammar formalism based on Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). From TDG, it inherits declarative word order constraints, the ability to distinguish multiple dimensions of linguistic description, and an axiomatization as a constraint satisfaction problem (Duchier, 2003) solvable using constraint programming (Apt, 2003). One of the benefits of this axiomatization is that the linear order of the words can be left underspecified, with the effect that the constraint solver can be applied for both parsing and generation.

XDG solving is efficient at least for the smaller-scale example grammars tested so far, but these good results hinge substantially on the assump-

tion of a 1:1-correspondence between words and nodes. As XDG has been created to cover not only syntax but also semantics, we have no choice but to weaken the 1:1-correspondence.

In this paper, we outline a way to break out of the 1:1-straightjacket, without sacrificing the potential for efficient parsing and generation. We introduce a new layer of lexical organization called *groups* above the basic XDG lexicon, allowing us to describe MWEs as tuples of dependency subgraphs. Groups can be compiled into simple lexical entries, which can then be used by the already existing XDG solver for parsing and generation. With groups, we can omit nodes present in the syntactic dimensions in the semantic dimensions, and thus get away from the 1:1-correspondence.

Groups are motivated by the *continuity hypothesis* of (Kay and Fillmore, 1999), assuming the continuity of the lexicon and the construction. They can also be regarded as a declarative formalization of Mel'čuk's paraphrasing rules, or as a realization of the *extended domain of locality* of TAG (Joshi, 1987) in terms of dependency grammar, as also proposed in (Nasr, 1996) and (Joshi and Rambow, 2003).

The structure of this paper is as follows. We introduce XDG in section 2. In section 3, we introduce groups, the new layer of lexical organization required for the modeling of MWEs, and in section 4, we show how to compile groups into simple lexical entries amenable for parsing and generation. Section 5 concludes the paper and outlines future work.

2 XDG

In this section, we explain the intuitions behind XDG, before proceeding with its formalization, and a description of the XDG solver for parsing and generation.

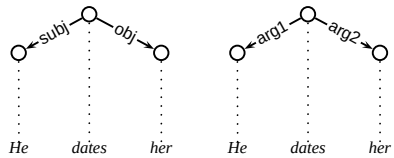
2.1 XDG intuitions

Extensible Dependency Grammar (XDG) is a new grammar formalism generalizing Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). XDG characterizes linguistic structure along arbitrary many *dimensions* of description. All dimensions correspond to labeled graphs, sharing the same set of nodes but having different edges.

The well-formedness conditions for XDG analyses are determined by *principles*. Principles can either be *one-dimensional*, applying to a single dimension only, or *multi-dimensional*, constraining the relation of several dimensions. Basic one-dimensional principles are treeness and valency. Multi-dimensional principles include climbing (as in (Duchier and Debusmann, 2001); one dimension must be a flattening of another) and linking (e.g. to specify how semantic arguments must be realized syntactically).

The *lexicon* plays a central role in XDG. For each node, it provides a set of possible lexical entries (feature structures) serving as the parameters for the principles. Because a lexical entry constrains all dimensions simultaneously, it can also help to synchronize the various dimensions, e.g. with respect to valency. For instance, a lexical entry could synchronize syntactic and semantic dimensions by requiring a subject in the syntax, and an agent in the semantics.

As an example, we show in (1) an analysis for the sentence *He dates her*, along two dimensions of description, immediate dominance (ID) and predicate-argument structure (PA). We display the ID part of the analysis on the left, and the PA part on the right:[1]



(1)

The set of edge labels on the ID dimension includes subj for subject and obj for object. On the PA dimension, we have arg1 and arg2 standing for the argument slots of semantic predicates.[2] The ID part of the analysis states that *He* is the subject, and *her* the object of *dates*. The PA part states that *He* is the first argument and *her* the second argument of *dates*.

[1] For lack of space, we omit the dimension of linear precedence (LP) from the presentation in this paper. For this dimension, we use the same theory as for TDG (Duchier and Debusmann, 2001).

[2] We could also use some set of thematic roles for the PA edge labels, but since the assumption of thematic roles is very controversial, we decided to choose more neutral labels.

The principles of the underlying grammar require that the ID part of each analysis is a tree, and the PA part a directed acyclic graph (dag).[3][4] In addition, we employ the valency principle on both dimensions, specifying the licensed incoming and outgoing edges of each node. The only employed multi-dimensional principle is the linking principle, specifying how semantic arguments are realized syntactically.

Figure 1 shows the lexicon of the underlying grammar. Each lexical entry corresponds to both a word and a semantic literal. $\mathsf{in}_{\mathrm{ID}}$ and $\mathsf{out}_{\mathrm{ID}}$ parametrize the valency principle on the ID dimension. $\mathsf{in}_{\mathrm{ID}}$ specifies the licensed incoming, and $\mathsf{out}_{\mathrm{ID}}$ the licensed outgoing edges. E.g. *He* licenses zero or one incoming edges labeled subj, and no outgoing edges. $\mathsf{in}_{\mathrm{PA}}$ and $\mathsf{out}_{\mathrm{PA}}$ parametrize the valency principle on the PA dimension. E.g. *dates* licenses no incoming edges, and requires precisely one outgoing edge labeled arg1 and one labeled arg2. link parametrizes the multi-dimensional linking principle. E.g. *dates* syntactically realizes its first argument by a subject, the second argument by an object.

Observe that all the principles are satisfied in (1), and hence the analysis is well-formed. Also notice that we can use the same grammar and lexicon for both parsing (from words) and generation (from semantic literals).

2.2 XDG formalization

Formally, an XDG grammar is built up of dimensions, a lexicon and principle, and characterizes a set of well-formed analyses.

A *dimension* is a tuple $D = (\mathit{Lab}, \mathit{Fea}, \mathit{Val}, \mathit{Pri})$ of a set *Lab* of edge labels, a set *Fea* of features, a set *Val* of feature values, and a set of one-dimensional principles *Pri*. A *lexicon* for the dimension D is a set $\mathit{Lex} \subseteq \mathit{Fea} \to \mathit{Val}$ of total feature assignments called lexical entries. An analysis on dimension D is a triple (V, E, F) of a set V of nodes, a set $E \subseteq V \times V \times \mathit{Lab}$ of directed labeled edges, and an assignment $F : V \to (\mathit{Fea} \to \mathit{Val})$ of lexical entries to nodes. V and E form a graph. We write Ana_D for the set of all possible analyses on dimension D. The principles characterize subsets of Ana_D. We assume that the elements of *Pri* are finite representations of such subsets.

An *XDG grammar* $((\mathit{Lab}_i, \mathit{Fea}_i, \mathit{Val}_i, \mathit{Pri}_i)_{i=1}^n, \mathit{Pri}, \mathit{Lex})$ consists of n dimensions, multi-dimensional principles *Pri*, and a lexicon *Lex*. An *XDG analysis* $(V, E_i, F_i)_{i=1}^n$ is an element of $\mathit{Ana} = \mathit{Ana}_1 \times \cdots \times \mathit{Ana}_n$ where all dimensions share the same set of nodes V. We call a dimension of a grammar *grammar dimension*.

Multi-dimensional principles specify subsets of *Ana*, i.e. of tuples of analyses for the individual dimensions. The lexicon $\mathit{Lex} \subseteq \mathit{Lex}_1 \times \cdots \times \mathit{Lex}_n$ constrains all dimensions at once, thereby synchronizing them. An XDG analysis is licensed by *Lex* iff $(F_1(v), \ldots, F_n(v)) \in \mathit{Lex}$ for every node $v \in V$.

In order to compute analyses for a given input, we employ a set of *input constraints* (*Inp*), which again specify a subset of *Ana*. XDG solving then amounts to finding elements of *Ana* that are licensed by *Lex*, and consistent with *Inp* and *Pri*. The input constraints e.g. determine whether XDG solving is to be used for parsing or generation. For parsing, they specify a sequence of words, and for generation, a multiset of semantic literals.

2.3 XDG solver

XDG solving has a natural reading as a constraint satisfaction problem (CSP) on finite sets of integers, where well-formed analyses correspond to the solutions of the CSP (Duchier, 2003). We have implemented an XDG solver (Debusmann, 2003) using the Mozart-Oz programming system (Mozart Consortium, 2004).

XDG solving operates on all dimensions concurrently. This means that the solver can infer information about one dimension from information on another, if there is either a multi-dimensional principle linking the two dimensions, or by the synchronization induced by the lexical entries. For instance, not only can syntactic information trigger inferences in syntax, but also vice versa.

Because XDG allows us to write grammars with completely free word order, XDG solving is an NP-complete problem (Koller and Striegnitz,

[3] In the following, we will call the ID part of an analysis ID tree, and the PA part PA dag.

[4] The PA structure is a dag and not a tree because we want it to reflect the re-entrancy e.g. in control constructions, where the same subject is shared by more than one node.

word	literal	in_{ID}	out_{ID}	in_{PA}	out_{PA}	link
He	*he'*	{subj?}	{}	{arg1?, arg2?}	{}	{}
dates	*date'*	{}	{subj!, obj!}	{}	{arg1!, arg2!}	{arg1 ↦ {subj}, arg2 ↦ {obj}}
her	*she'*	{obj?}	{}	{arg1?, arg2?}	{}	{}

Figure 1: Lexicon for the sentence *He dates her.*

2002). This means that the worst-case complexity of the solver is exponential. The average-case complexity of many smaller-scale grammars that we have experimented with seems polynomial, but it remains to be seen whether we can scale this up to large-scale grammars.

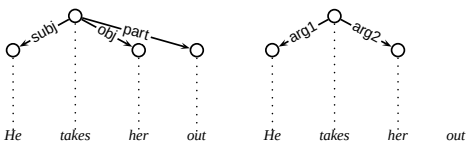
3 Groups

In this section, we consider MWE paraphrases and propose to model them as tuples of dependency subgraphs called *groups*. We start with an example: Consider (3), a paraphrase of (2):

He dates her. (2)

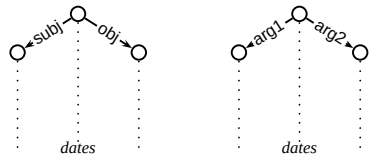
He takes her out. (3)

We display the XDG analysis of (3) in (4). Again, the ID tree is on the left, and the PA dag on the right:[5]

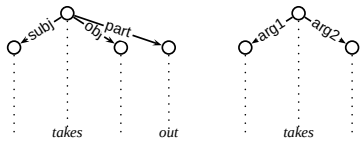


(4)

This example demonstrates that we cannot simply treat MWEs as contiguous word strings and include those in the lexicon, since the MWE *takes out* is interrupted by the object *her* in (3). Instead, we choose to implement the *continuity hypothesis* of (Kay and Fillmore, 1999) in terms of DG, and model MWEs as dependency subgraphs. We realize this idea by a new layer of lexical organization called *groups*. A group is tuple of dependency subgraphs covering one or more nodes, where each component of the tuple corresponds to a grammar dimension. We call a group component *group dimension*. We display the group corresponding to *dates* in (5), and the one corresponding to *takes out* in (6).



(5)



(6)

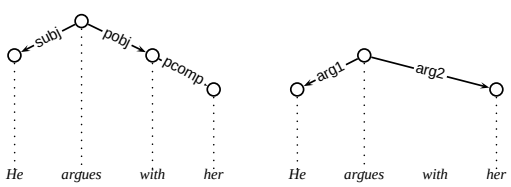
Groups can leave out nodes present in the ID dimension on the PA dimension. E.g. in (6), the node corresponding to the particle *out* is present on the ID dimension, but left out on the PA dimension. In this way, groups help us to weaken the 1:1-correspondence between words and nodes.

We can make use of groups also to handle more complicated MWE paraphrases. Consider (8) below, a support verb construction paraphrasing (7):

He argues with her. (7)

He has an argument with her. (8)

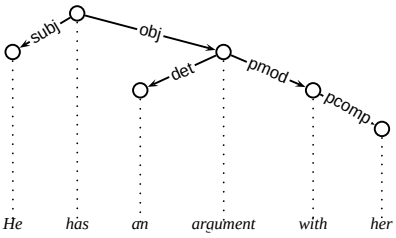
In (8), *has* is only a support verb; the semantic head of the construction is the noun *argument*. We display the ID tree and PA dag of (7) in (9). The ID tree of (8) is in (10), and the PA dag in (11):[6]



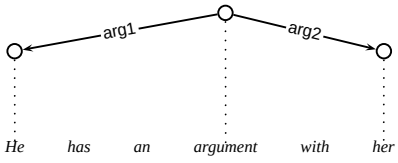
(9)

[5] In the ID tree, part stands for a particle.

[6] In the ID trees, pobj stands for a prepositional object, pcomp for the complement of a preposition, pmod for a prepositional modifier, and det for a determiner.

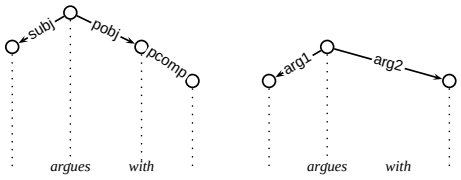


(10)

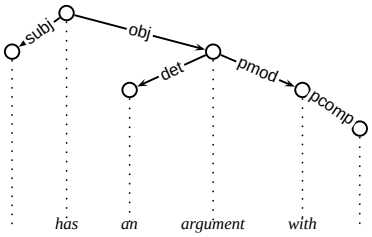


(11)

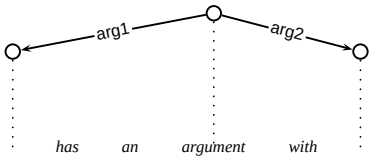
In (9), the node corresponding to *with* is deleted in the PA dag. In (11), the support verb construction leads to the deletion of three nodes (corresponding to resp. *has*, *an* and *with*). These deletions are reflected in the corresponding groups. The group corresponding to *argues with* is displayed in (12), and the group corresponding to *has an argument with* in (13) (ID) and (14) (PA):



(12)



(13)



(14)

Groups can capture difficult constructions such as the support verb construction above in an elegant and transparent way, without having to resort to complicated machinery. The key aspect of groups is their multi-dimensionality, describing tuples of dependency subgraphs spanning over a shared set of nodes. This sharing enables groups to express interdependencies between the different dimensions. For instance in (13) and (14), the noun *argument*, the object of the support verb *has* in the ID dimension, is the semantic head in the PA dimension.

4 Compilation

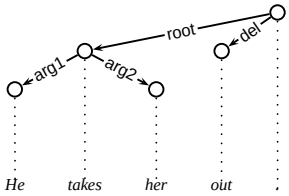
In this section, we show how to compile groups into simple lexical entries. The benefit of this is that we can retain XDG in its entirety, i.e. we can retain its formalization and its axiomatization as a constraint satisfaction problem. This means we can also retain the implementation of the XDG solver, and continue to use it for parsing and generation.

4.1 Node deletion

The 1:1-correspondence between words and nodes is a key assumption of XDG. It requires that on each dimension, each word must correspond to precisely one node in the dependency graph. The groups shown in the previous section clearly violate this assumption as they allow nodes present on the ID dimension to be omitted on the PA dimension.

The first step of the compilation aims to accommodate this deletion of nodes. To this end, we assume for each analysis an additional root node, e.g. corresponding to the full stop at the end of the sentence. Each root in an XDG dependency graph becomes a daughter of this additional root node, the edge labeled root. The trick for deleting nodes is now the following: Each deleted node also becomes a daughter of the additional root node, but with a special node label indicating that this node is regarded to be deleted (del).

As an example, here is the PA dag for example (3) including the additional root node:



(15)

4.2 Dependency subgraphs

The second step of the compilation is to compile the dependency subgraphs into lexical entries for individual words. To this end, we make use of the valency principle. For each edge from v to v' labeled l within a group, we require that v has an outgoing edge labeled l, and that v' licenses an incoming edge labeled l. I.e. we include $l!$ in the out specification of the lexical entry for v, and $l?$ in the in specification of the lexical entry for v'.

4.3 Group coherence

The final step of the compilation is about ensuring *group coherence*, i.e. to ensure that the inner nodes of a group dimension (neither the root nor the leaves) stay within this group in the dependency analysis. In other words, group coherence make sure that the inner nodes of a group dimension cannot become daughters of nodes of a different group. In our support verb example, group coherence ensures that e.g. that the determiner *an* cannot become the determiner of a noun of another group. We realize this idea through a new XDG principle called *group coherence principle*. This principle must hold on all dimensions of the grammar.

Given a set of group identifiers $Group$, the principle assumes two new features: $\mathsf{group} : Group$, and $\mathsf{outgroups} : Lab \to 2^{Group}$. For each node v, $\mathsf{group}(v)$ denotes the group identifier of the group of v. For each edge within a group from v to v' labeled l, i.e. for each edge to an inner node, $\mathsf{outgroups}(v)(l)$ denotes the singleton set containing the group of both v and v'. For each edge from v labeled l which goes outside of a group, $\mathsf{outgroups}(v)(l) = Group$, i.e. the edge can end in any other group. We formalize the group coherence principle as follows, where $v \xrightarrow{l} v'$ denotes an edge from v to v' labeled l:

$$v \xrightarrow{l} v' \quad \Rightarrow \quad \mathsf{group}(v') \in \mathsf{outgroups}(v)(l) \tag{16}$$

4.4 Examples

For illustration, we display the lexical entries of two compiled groups: The group *argues with* (with identifier g1), and the group *has an argument with* (with identifier g2), resp. in Figure 2 and Figure 3. We omit the specification of outgroups for the PA dimension for lack of space, and since it is not relevant for the example: In all groups, there are no edges which stay within a group in the PA dimension.

4.5 Parsing and generation

We can use the same group lexicon for parsing and generation, but we have to slightly adapt the compilation for the generation case.

For parsing, we can use the XDG parser as before, without any changes.

For generation, we assume a set Sem of semantic literals, multisets of which must be verbalized. To do this, we assume a function $\mathsf{groups} : Sem \to 2^{Group}$, mapping each semantic literal to a set of groups which verbalize it.

Now before using the XDG solver for generation, we have to make sure for each semantic literal *s* to be verbalized that the groups which can potentially verbalize it have the same number of nodes. For this, we calculate the maximum number n of syntactic nodes for the groups assigned to *s*, and fill up the groups having less syntactic nodes with deleted nodes. Then for XDG solving, we introduce precisely n nodes for literal *s*. Using groups for generation thus comes at the expense of having to introduce additional nodes.[7]

As an example, consider we want to verbalize the semantic literal *argue*. $\mathsf{groups}(\textit{argue}) = \{\mathsf{g1}, \mathsf{g2}\}$, i.e. either groups g1 or g2 can verbalize it. The maximum number n of syntactic nodes for the two groups is 4 for g2 (*has an argument with*). Hence, we have to fill up the groups having less syntactic nodes with deleted nodes. In this case, we have to add two deleted nodes to the group g1 (*argue with*) to get to four nodes. The resulting lexical entries encoding g1 are displayed in Figure 4. The lexical entries for g2 stay the same as in Figure 3.

After this step is done, we can use the existing XDG solver to generate from the semantic literals precisely the two possible verbalizations (7) and (8), as intended.

[7] We should emphasize that n is not at all unrestricted. For each semantic literal *s* to be verbalized, we have to introduce only as many nodes as are contained in the largest group which can verbalize *s*.

word	literal	group	$\text{outgroups}_{\text{ID}}$	in_{ID}	out_{ID}	in_{PA}	out_{PA}	link
argues	*argue'*	g1	$\{\text{pobj} \mapsto \{\text{g1}\}\}$	{root?}	{subj!, pobj!}	{root?}	{arg1, arg2}	$\{\begin{smallmatrix}\text{arg1} \mapsto \{\text{subj}\}\\ \text{arg2} \mapsto \{\text{pcomp}\}\end{smallmatrix}\}$
with	*argue'*	g1	{}	{pobj?}	{pcomp!}	{del?}	{}	{}

Figure 2: Lexical entries encoding the group for *argues with*

word	literal	group	$\text{outgroups}_{\text{ID}}$	in_{ID}	out_{ID}	in_{PA}	out_{PA}	link
has	*argue'*	g2	$\{\text{obj} \mapsto \{\text{g2}\}\}$	{root?}	{subj!, obj!}	{del?}	{}	{}
an	*argue'*	g2	{}	{det?}	{}	{del?}	{}	{}
argument	*argue'*	g2	$\{\begin{smallmatrix}\text{det} \mapsto \{\text{g2}\}\\ \text{pmod} \mapsto \{\text{g2}\}\end{smallmatrix}\}$	{obj?}	{det!, pmod!}	{root?}	{arg1!, arg2!}	$\{\begin{smallmatrix}\text{arg1} \mapsto \{\text{subj}\}\\ \text{arg2} \mapsto \{\text{pcomp}\}\end{smallmatrix}\}$
with	*argue'*	g2	{}	{pmod?}	{pcomp!}	{del?}	{}	{}

Figure 3: Lexical entries encoding the group for *has an argument with*

5 Conclusion

We extended the XDG grammar formalism with a means to break out of the straightjacket of the 1:1-correspondence between words and nodes. To this end, we proposed the new notion of *groups*, allowing to enrich the XDG lexicon with tuples of dependency subgraphs. We illustrated how to tackle complicated MWEs such as support verb constructions with this new idea, and how to compile groups into simple lexical entries.

We see two main benefits of our approach. The first is that we can retain the XDG formalization, and also its axiomatization as a constraint satisfaction problem, in its entirety. Thus, we can simply continue to use the existing XDG solver for parsing and generation. The second benefit is that we can use the same group lexicon for both parsing and generation, the only difference being that for generation, we have to slightly adapt the compilation into lexical entries.

There are many issues which have remained untouched in this paper. For one, we did not talk about our treatment of word order in this paper for lack of space. Word order is among the best researched issues in the context of XDG. For a thorough discussion about word order in XDG, we refer to (Duchier and Debusmann, 2001) and (Debusmann, 2001).

Another issue is that of the relation between groups and the meta-grammatical functionality of the XDG lexicon, offering lexical inheritance, templates and also disjunction in the sense of *crossings* (Candito, 1996) to lexically state linguistic generalizations. How well groups can be integrated with this meta-grammatical functionality is an open question.

XDG research has so far mainly been focused on parsing, only to a very small extent on generation, and to no extent at all on Machine Translation (MT). There is still a lot to do on both of the latter topics, and even for parsing, we are only at the beginning. Although with our smaller-scale example grammars, parsing and generation takes polynomial time, we have yet to find out how we can scale this up to large-scale grammars. We have started importing and inducing large-scale grammars from existing resources, but can so far only speculate about if and how we can parse them efficiently. In a related line of research, we are also working on the incorporation of statistical information (Dienes et al., 2003) to help us to guide the search for solutions. This could improve performance because we would only have to search for a few good analyses instead of enumerating hundreds of them.

Acknowledgements

The idea for groups and this paper was triggered by three events in fall 2003. The first was a workshop where Peter Dienes, Stefan Thater and me worked out how to do generation with XDG. The second was a presentation on the same workshop by Aravind Joshi and Owen Rambow of their encoding of DG in TAG, and the third was a talk by Charles Fillmore titled *Multiword Expressions: An Extremist Approach*. I'd like to thank all of them. And I'd like to thank all the others involved with XDG, in alphabetical order: Ondrej Bojar, Denys Duchier, Alexander Koller, Geert-Jan Krui-

word	literal	group	$\text{outgroups}_{\text{ID}}$	in_{ID}	out_{ID}	in_{PA}	out_{PA}	link
argues	*argue'*	g1	{pobj ↦ {g1}}	{root?}	{subj!, pobj!}	{root?}	{arg1, arg2}	{ arg1 ↦ {subj} arg2 ↦ {pcomp} }
with	*argue'*	g1	{}	{pobj?}	{pcomp!}	{del?}	{}	{}
ϵ	*argue'*	g1	{}	{del?}	{}	{del?}	{}	{}
ϵ	*argue'*	g1	{}	{del?}	{}	{del?}	{}	{}

Figure 4: Lexical entries for generation, encoding the group for *argues with*

jff, Vladislav Kubon, Marco Kuhlmann, Joachim Niehren, Martin Platek and Gert Smolka for their support and many helpful discussions.

References

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

Norbert Bröker. 1999. *Eine Dependenzgrammatik zur Kopplung heterogener Wissensquellen*. Linguistische Arbeiten 405. Max Niemeyer Verlag, Tübingen/GER.

Marie-H`eléne Candito. 1996. A principle-based hierarchical representation of LTAG. In *Proceedings of COLING 1996*, Kopenhagen/DEN.

Ralph Debusmann. 2001. A declarative grammar formalism for dependency grammar. Master's thesis, University of Saarland.

Ralph Debusmann. 2003. A parser system for extensible dependency grammar. In Denys Duchier, editor, *Prospects and Advances in the Syntax/Semantics Interface*, pages 103–106. LORIA, Nancy/FRA.

Peter Dienes, Alexander Koller, and Marco Kuhlmann. 2003. Statistical A* Dependency Parsing. In *Prospects and Advances in the Syntax/Semantics Interface*, Nancy/FRA.

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

Denys Duchier. 2003. Confi guration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation*, 1(3–4):307–336.

Kim Gerdes and Sylvain Kahane. 2001. Word order in german: A formal dependency grammar using a topological hierarchy. In *Proceedings of ACL 2001*, Toulouse/FRA.

Johannes Heinecke, Jürgen Kunze, Wolfgang Menzel, and Ingo Schröder. 1998. Eliminative parsing with graded constraints. In *Proceedings of COLING/ACL 1998*, pages 526–530, Montreal/CAN.

Aravind Joshi and Owen Rambow. 2003. A formalism for dependency grammar based on tree adjoining grammar. In *Proceedings of MTT 2003*, pages 207–216, Paris/FRA.

Aravind K. Joshi. 1987. An introduction to tree-adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam/NL.

Paul Kay and Charles J. Fillmore. 1999. Grammatical constructions and linguistic generalizations: the what's x doing y? construction. *Language*, 75:1–33.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of ACL 2002*, Philadelphia/USA.

Geert-Jan M. Kruijff. 2001. *A Categorial-Modal Architecture of Informativity*. Ph.D. thesis, Charles University, Prague/CZ.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State Univ. Press of New York, Albany/USA.

Igor Mel'čuk. 1996. Lexical functions: a tool for the description of lexical relations in a lexicon. In Leo Wanner, editor, *Lexical Functions in Lexicography and Natural Language Processing*. John Benjamins.

Mozart Consortium. 2004. The Mozart-Oz website. http://www.mozart-oz.org/.

Alexis Nasr. 1995. A formalism and a parser for lexicalised dependency grammars. In *4th International Workshop on Parsing Technologies*, pages 186–195, Prague/CZ.

Alexis Nasr. 1996. *Un système de reformulation automatique de phrases fondé sur la Théorie Sens-Texte: application aux langues contrôlées*. Ph.D. thesis, Université Paris 7.

Petr Sgall, Eva Hajicova, and Jarmila Panevova. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. D. Reidel, Dordrecht/NL.

Lucien Tesni`ere. 1959. *Éléments de Syntaxe Structurale*. Klincksiek, Paris/FRA.