A Parser System for Extensible Dependency Grammar

Ralph Debusmann
Programming Systems Lab
Universität des Saarlandes, Geb. 45
Postfach 15 11 50
66041 Saarbrücken, Germany
rade@ps.uni-sb.de

Abstract

This paper introduces a parser system for the meta grammar formalism of Extensible Dependency Grammar (XDG). XDG is a generalisation of Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). The XDG parser system comprises a constraint-based parser for all possible instances of XDG, a statically typed grammar input language, and a flexible backend for handling parser output. A powerful graphical user interface provides for easy accessibility of all the functionality of the system. In the future, we will use the XDG parser system to accomodate new dependency grammar formalisms such as Semantic Topological Dependency Grammar (STDG), and to experiment with other interesting XDG instances.

1 Introduction

Extensible Dependency Grammar (XDG) is a new meta grammar formalism for dependency grammar (Tesnière, 1959). XDG is based on the grammar formalism of Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). XDG can be likened to Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994): both XDG and HPSG are meta grammar formalisms which need to be instantiated with particular principles and parameters to obtain specific grammar formalisms. The fundamental difference is that whereas HPSG is based on typed feature structures, XDG is based on graph descriptions.

A parser system for TDG can be found in (Duchier and Debusmann, 2002). The parser uses constraint-based techniques developed in (Duchier, 1999) and (Duchier, 2002) to achieve polynomial parse time in the average-case, although the TDG parsing problem has proven to be NP-complete (Koller and Striegnitz, 2002) in the worst-case. The TDG parser system is written using the Mozart-Oz programming language (Moz, 1998).

In this paper, we present a parser system for XDG. The system includes a constraint-based parser for all possible instances of XDG, a statically typed grammar input language, and a flexible backend for handling parser output. The XDG parser is a generalisation of the TDG parser, and achieves the same polynomial parse time in the average case for the TDG instance. The XDG parser system is also written in Mozart-Oz.

The XDG parser system has at least two purposes. On the one hand, we use it for parsing grammars written in Semantic Topological Dependency Grammar (STDG), an extension of TDG with a syntax-semantics interface to underspecified semantics in the Constraint Language for Lambda Structures (CLLS) (Egg et al., 1998). On the other, we provide the parser system as a tool for research on various interesting dependency grammar formalisms which are instantiations of XDG.

The structure of the paper is as follows: we briefly introduce XDG in section 2. In section 3, we present the XDG parser, and proceed to present the frontend and backend of the system in sections 4 and 5. We introduce the graphical user interface (GUI) of the system in section 6, before we conclude in section 7.

2 Extensible Dependency Grammar (XDG)

XDG is a graph description language for a set of *graph dimensions*, where each graph dimension corresponds to a graph. All graphs share the same set of nodes, but have different edges and different edge labels. Simple feature structures can be attached to each node, and by lexicalisation, XDG can be turned into a powerful dependency-based meta grammar formalism.

The well-formedness conditions of an XDG analysis are stipulated by parametrised *principles*. Each dimension uses a set of principles, stipulating restrictions on the licensed graphs on this dimension. Principles can also pose restrictions on any number of dimensions simultaneously. An XDG analysis is well-formed only if all used principles are satisfied on all dimensions. Principles are taken from a shared and extensible *principle library*.

The recipe to get an instance of XDG is easy: 1) define the set of used graph dimensions, and 2) define the set of used principles (and their parameters). The TDG instance uses two dimensions, ID (Immediate Dominance) and LP (Linear Precedence). On the ID dimension, we use the tree, in (accepted edges) and out (valency) principles. On the LP dimension, we use the tree, in, out, projectivity, climbing and barriers principles. A detailed description of these principles can be found in (Duchier, 2002).

3 XDG parser

The XDG parser is at the heart of the XDG parser system. It can be instantiated to yield parsers for any instance of XDG, including TDG. The parser is a generalisation of the TDG parser (Duchier and Debusmann, 2002), and makes use of the same programming techniques, originally developed in (Duchier, 1999) and (Duchier, 2002).

The XDG parser can handle any number of dimensions, and any number of principles used on them. It provides a predefined *principle library*, including implementations of all the principles required for the TDG instance of XDG. The principle library can be freely extended with new principles written in Mozart-Oz.

Each principle in the principle library consists of a number of Oz functors which implement the principle. Therefore, it is not only possible to replace whole principles with new ones, but also to only partially substitute parts of principles with others.[1]

The parser retains the good average-case performance of the TDG parser for the TDG instance and small test grammars. As with the TDG parser, the lack of bigger grammars has as of yet prohibited proper evaluation. However, we are currently working on the creation of bigger grammars for Czech, English and German.

4 Frontend

The XDG parser system frontend provides a input language for grammars called IL (Intermediate Language). A grammar defines 1) an instance of XDG (i.e. a set of used dimensions and a set of used principles and parameters), and 2) a lexicon. The IL is statically typed to ease the discovery of errors.

Grammars written in the IL are hard to read for humans. This is why the parser system arranges for the definition of input lanuguages on top of the IL. At the moment, the system provides a language called UL (User Language) which is much better suited to write grammars in. We also provide an emacs mode for the UL. In the near future, we will provide an XML-based input language to improve interoperability.

Grammars written in any language are first compiled into the IL, and then into the PL (Parser Language), which is the language used in the XDG parser itself.[2]

Also new to the XDG parser system is the inclusion of features originating from Metagrammar (Candito, 1996). Most notably, it is possible to use disjunction almost everywhere in the lexicon to specify a non-deterministic choice. Together with conjunction (which amounts to lexical inheritance

[1] This is particularly useful for non-deterministic search: the existing functor implementing naive search can easily be replaced by a functor implementing oracle-guided best-first search.

[2] At this stage, one big advantage of the new XDG parser system over the old TDG parser system is the ability to save precompiled grammars to disk, such that there is no need to compile the grammars again and again before use.

of lexical types), the frontend provides for the succinct description of a number of lexical entries by just one metagrammatical expression. Lexical economy is further improved by parametrised lexical types. Below is an example metagrammatical expression written in the User Language UL:

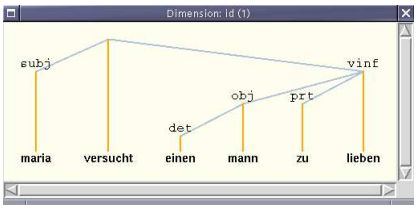
(1)
$$\texttt{finite}(\$\ \texttt{third}\ \texttt{sg})\ \&\ (\texttt{canonical} \mid \texttt{noncanonical})$$

The expression uses three lexical types (`finite`, `canonical` and `noncanonical`). `finite` is parametrised by an agreement expression (here: `$ third sg` for *third person singular*), and stands for a finite verb. `canonical` and `noncanonical` stand for *canonical position* and *non-canonical position* respectively. The expression defines a set of lexical entries which are finite verbs in third person singular inflection, and which are in canonical position or in non-canonical position. After compilation, this yields at least two distinct lexical entries, one for the finite verb in canonical position, and one in non-canonical position. Both entries will have third person singular inflection.

5 Backend

After parsing, a parser system needs to provide a means to conveniently display the parsing result. This is the role of the backend of the XDG parser system. We provide an *output library* of output functors to display the result of a parse either graphically or textually. It is also possible to get output which is directly usable in LaTeX documents. Below in (2), we show an example graph output:

(2)

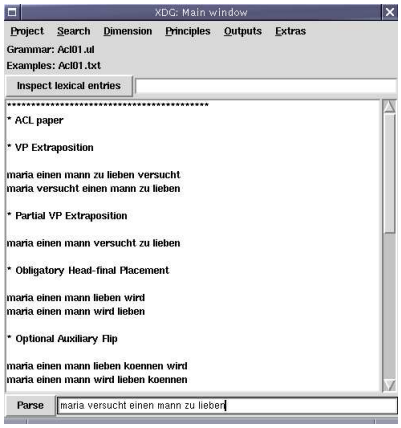


The output library is also extensible. We provide means to easily write own output functors in Mozart-Oz, to display both partial and full parses.

6 GUI

The GUI of the XDG parser system makes all functionality easily accessible and simplifies grammar debugging. It allows to precompile grammars, load them and create parsing log files. Grammars can be loaded in any of the available input languages, and all available output functors of the backend can be used. We display an example screenshot in (3) below:

(3)



7 Conclusions

We introduced a new parser system for Extensible Dependency Grammar (XDG), as a generalisation of the existing TDG parser system (Duchier and Debusmann, 2002). The XDG parser system can be found in (Debusmann and Duchier, 2003). It improves in almost all respects on the TDG parser system: the XDG parser per se is much more flexible and extensible, and can cater for all instances of the XDG meta grammar formalism. The frontend is also much more flexible and includes features from Metagrammar to significantly improve lexical economy. The backend is much more flexible too, and also easily extensible.

We hope that the XDG parser system encourages many people to join working on XDG and its various instances.

References

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

Ralph Debusmann and Denys Duchier. 2003. eXtensible dependency grammar. http://www.mozart-oz.org/mogul/info/debusmann/xdg.html.

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

Denys Duchier and Ralph Debusmann. 2002. Topological Dependency Grammar 1.2. http://www.mozart-oz.org/mogul/info/duchier/coli/dg.html.

Denys Duchier. 1999. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on the Mathematics of Language*, Orlando/FL.

Denys Duchier. 2002. Configuration of labeled trees under lexicalized constraints and principles. To appear in the Journal of Language and Computation.

Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. 1998. Constraints over lambda-structures in semantic underspecification. In *Proceedings of COLING/ACL 1998*, pages 353–359, Montreal/CAN.

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

1998. Mozart. http://www.mozart-oz.org/.

Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Lucien Tesnière. 1959. *Eléments de Syntaxe Structurale*.