The XDG Development Kit

Ralph Debusmann

Programming Systems Lab, Saarbrücken

IGK Colloquium, December 16th, 2004

Overview

Overview

Declarative Grammar Formalisms

- specify linguistic knowledge independently from processing
- parsers/generators: can be generically created for all grammars

Examples

- LFG (Bresnan 2001)
- HPSG (Pollard/Sag 1994)
- TAG (Joshi 1987)
- CCG (Steedman 2000)

Declarative Grammar Formalisms

- specify linguistic knowledge independently from processing
- parsers/generators: can be generically created for all grammars

Examples

- LFG (Bresnan 2001)
- HPSG (Pollard/Sag 1994)
- TAG (Joshi 1987)
- CCG (Steedman 2000)

Grammar Development Systems

- tools for grammar creation
- parsers
- generators

Examples

- XLE (Kaplan/Maxwell 1996) for LFG
- LKB (Copestake 2002) for HSPG
- XTAG (XTAG Group 2001) for TAG
- OpenCCG (White 2004) for CCG

Grammar Development Systems

- tools for grammar creation
- parsers
- generators

Examples

- XLE (Kaplan/Maxwell 1996) for LFG
- LKB (Copestake 2002) for HSPG
- XTAG (XTAG Group 2001) for TAG
- OpenCCG (White 2004) for CCG

Existing Formalisms and Word Order

- languages with freer word order than English (e.g. German, Czech, Hindi etc.) pose problems
- Smolka (Smolka/Uszkoreit 1996): Could more advanced constraint programming techniques improve linguistic processing?

Axiomatization of Dependency Trees

- (Duchier 1999, 2003): axiomatization of valid dependency trees using finite set constraints
- parsing: reduced to constraint programming
- grammar formalism: Topological Dependency Grammar (TDG) (Duchier/Debusmann 2001)
- elegant new treatment of free word order

Extensible Dependency Grammar (XDG)

- generalization of TDG (Debusmann et al. 2004)
- graph description language for modeling arbitrary many levels of linguistic structure
- same parsing methods by constraint programming (Duchier 1999, 2003)
- goes beyond syntax:
  - semantics (Debusmann/Duchier/Koller/Kuhlmann/Smolka/Thater 2004)
  - information structure (Debusmann/Postolache/Traat 2005), out of an IGK-project also with Ciprian Gerstenberger and Stefan Thater

XDG Grammar Development Kit (XDK)

- first grammar development system for XDG
- implemented in Mozart/Oz, published in the Mozart Global User Library (MOGUL)

Facilities

- new abstract lexicon language
- grammar file compiler
- graphical interfaces
- solver for parsing and generation

XDG Grammar Development Kit (XDK)

- first grammar development system for XDG
- implemented in Mozart/Oz, published in the Mozart Global User Library (MOGUL)

Facilities

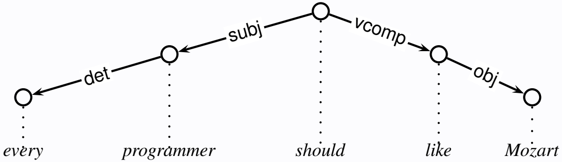
- new abstract lexicon language
- grammar file compiler
- graphical interfaces
- solver for parsing and generation

Overview

Graphs

- XDG describes labeled graphs
- uses the linguistic notion of dependency grammar

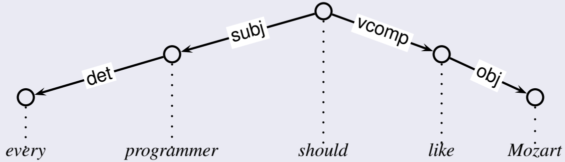
Example



Graphs

- XDG describes labeled graphs
- uses the linguistic notion of dependency grammar

Example

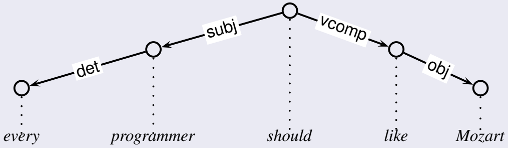


Multiple Graphs

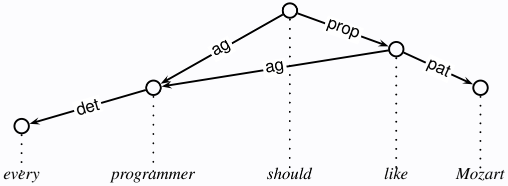
- XDG typically describes an arbitrary number of graphs called dimensions
- same set of nodes, different edges

Example

Syntax tree

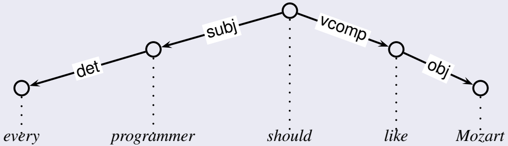


Semantic dag

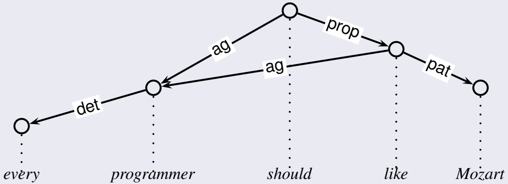


Example

Syntax tree



Semantic dag



Well-Formedness Conditions

- interaction of principles and the lexicon
- principles: restrictions on one or more dimensions
- subset of an extensible principle library
- lexicon: feature structures controlling the principles

Example Principles

- tree: dimension i must be a tree
- dag: dimension i must be a dag
- valency: for each node on dimension i, the incoming edges must be licensed by the in specification, and the outgoing edges by the out specification
- order: constrains the order of words on dimension i, e.g. subjects precede objects
- linking: constrains how arguments on dimension i (semantics) must be realized on dimension j (syntax), e.g. agents are realized as subjects

Example Lexical Entry

- lexical entry for *like*, controls valency and linking principles:

$$
\textit{like} = \begin{bmatrix} \textit{syn} & : & \begin{bmatrix} \text{in} & : & \{\text{vcomp?}\} \\ \text{out} & : & \{\text{obj!}\} \end{bmatrix} \\ \textit{sem} & : & \begin{bmatrix} \text{in} & : & \{\text{prop?}\} \\ \text{out} & : & \{\text{ag!}, \text{pat!}\} \\ \text{link} & : & \{\text{ag} : \{\text{subj}\} \\ & & \text{pat} : \{\text{obj}\}\} \end{bmatrix} \end{bmatrix}
$$

Overview

XDK Features

- new abstract lexicon language
- grammar file compiler
- graphical interfaces
- solver for parsing and generation

Lexicon Language

- XDG: linguistic information mostly specified in the lexicon
- but: lexicon grows huge even for medium-sized grammars
- need facilities for adequate modularization and factorization
- types: specify feature structures, define combination operation
- metagrammar (Crabbe/Duchier 2004): abstract description language for lexicon construction

Types

- each type: set L and partial function $\sqcap : L \times L \to L$ (combination operation of L)
- $\sqcap$: typically greatest lower bound

Supported types

domains, records, valencies, sets, tuples, strings

Types

- each type: set L and partial function $\sqcap : L \times L \to L$ (combination operation of L)
- $\sqcap$: typically greatest lower bound

Supported types

domains, records, valencies, sets, tuples, strings

Domain Types

- e.g. set of edge labels:

$$\textit{syn.label} \quad = \quad \{\mathsf{det}, \mathsf{subj}, \mathsf{obj}, \mathsf{vcomp}\}$$

- combination operation: $a \sqcap a = a$, $a \sqcap b$ undefined for $a \neq b$

Record Types

- given set of features $(f_i)_{i=1\ldots n}$ and types $T_i = (L_i, \sqcap_i)$:

$$\{f_1 : v_1, \ldots, f_n : v_n\}$$

  where $v_i \in L_i$.
- combination operation defined feature-wise:

$$\{f_1 : v_1, \ldots, f_n : v_n\} \sqcap \{f_1 : v'_1, \ldots, f_n : v'_n\} = \{f_1 : v_1 \sqcap_1 v'_1, \ldots, f_n : v_n \sqcap_n v'_n\}$$

Valency Types

- e.g. in and out specifications:

$$\mathit{syn.valency} \quad = \quad \mathsf{valency}(\mathit{syn.label})$$

- defines *syn.valency* to be the record type:

$$\{\mathit{det} : \mathit{mode},\ \mathit{subj} : \mathit{mode},\ \mathit{obj} : \mathit{mode},\ \mathit{vcomp} : \mathit{mode}\}$$

- $\mathit{mode} = \{0, ?, !, *\}$
- mode combination operation (specialization):

$$0 \sqcap x = x \qquad * \sqcap ! = ! \qquad * \sqcap ? = ? \qquad ? \sqcap ! = !$$

- convenient notation:

$$\{\mathit{det} : 0, \mathit{subj} : !, \mathit{obj} : ?, \mathit{vcomp} : 0\} \quad = \quad \{\mathsf{subj}!, \mathsf{obj}?\}$$

Metagrammar

- abstract description language for lexicon construction
- lexical classes:

$$\mathit{Class} \quad ::= \quad \mathit{ClassName} \rightarrow \mathit{ClassBody}$$

- class body:

$$\begin{array}{lll} \mathit{ClassBody} & ::= & \mathit{ClassBody}_1 \,\&\, \mathit{ClassBody}_2 \\ & | & \mathit{ClassBody}_1 \mid \mathit{ClassBody}_2 \\ & | & \mathit{ClassName} \\ & | & \mathit{partialLexicalEntry} \end{array}$$

- &: combination operation
- |: non-deterministic choice
- *ClassName*: class reference

Example (1)

- finite verbs can be roots or the head of a relative clause:

$$\begin{array}{rcl} \mathsf{root} & \rightarrow & \left[\ \mathit{syn}\ :\ \left[\ \mathsf{in}\ :\ \{\}\ \right]\ \right] \\ \mathsf{rel} & \rightarrow & \left[\ \mathit{syn}\ :\ \left[\ \mathsf{in}\ :\ \{\mathsf{relcl?}\}\ \right]\ \right] \\ \mathsf{finite} & \rightarrow & \mathsf{root} \mid \mathsf{rel} \end{array}$$

Example (2)

- finite verbs may be either intransitive, transitive or ditransitive:

$$
\begin{array}{rcl}
\text{intr} & \rightarrow & \left[\ \textit{syn} : \left[\ \text{out} : \{\text{subj!}\}\ \right]\ \right] \\
\text{tr} & \rightarrow & \text{intr}\ \&\left[\ \textit{syn} : \left[\ \text{out} : \{\text{obj!}\}\ \right]\ \right] \\
\text{ditr} & \rightarrow & \text{tr}\ \&\left[\ \textit{syn} : \left[\ \text{out} : \{\text{iobj!}\}\ \right]\ \right] \\
\text{verb} & \rightarrow & \text{intr} \mid \text{tr} \mid \text{ditr}
\end{array}
$$

Example (3)

- finite verb:

$$\begin{array}{rcl} \mathsf{finiteVerb} & \rightarrow & \mathsf{finite}\ \&\ \mathsf{verb} \\ & = & (\mathsf{root} \mid \mathsf{rel})\ \&\ (\mathsf{intr} \mid \mathsf{tr} \mid \mathsf{ditr}) \end{array}$$

- all possibilities of

$$\begin{array}{lll} (\mathsf{root}\ \&\ \mathsf{intr}) & (\mathsf{root}\ \&\ \mathsf{tr}) & (\mathsf{root}\ \&\ \mathsf{ditr}) \\ (\mathsf{rel}\ \&\ \mathsf{intr}) & (\mathsf{rel}\ \&\ \mathsf{tr}) & (\mathsf{rel}\ \&\ \mathsf{ditr}) \end{array}$$

- e.g.:

$$\mathsf{rel}\ \&\ \mathsf{ditr} \rightarrow \left[\ \mathit{syn} : \left[\begin{array}{rcl} \mathsf{in} & : & \{\mathsf{relcl?}\} \\ \mathsf{out} & : & \{\mathsf{subj!}, \mathsf{obj!}, \mathsf{iobj!}\} \end{array}\right]\ \right]$$

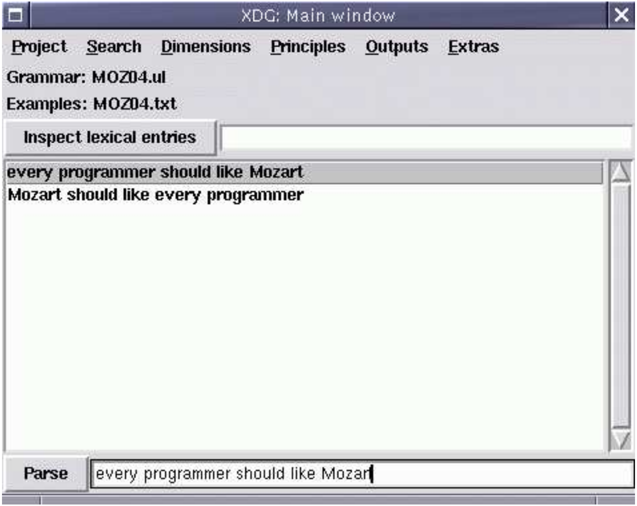
Grammar File Compiler

- fast static grammar checker
- fast grammar file compilation
- prepared for very large grammars (GNU GDBM support)
- three concrete syntaxes for different purposes:
  - XML language: automated grammar development
  - User Language (UL): handcrafted grammars
  - Intermediate Language (IL): record-based language, tailored for Mozart/Oz and further processing within the XDK

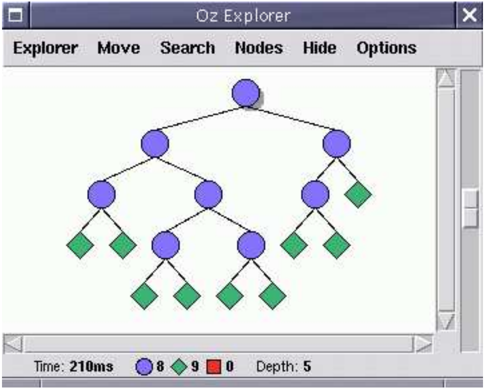
Graphical Interfaces

- comprehensive graphical user interface (GUI)
- solver search tree visualization: Oz Explorer (Schulte 1997), IOzSeF (Tack 2003)
- visualization of partial/full analyses: output library:
  - Tcl/Tk dag display
  - LaTeX dag output (using Denys Duchier's dtree.sty)
  - internal solver language output using the Oz Inspector (Brunklaus 2000)

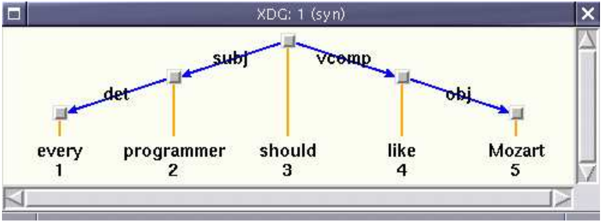
GUI

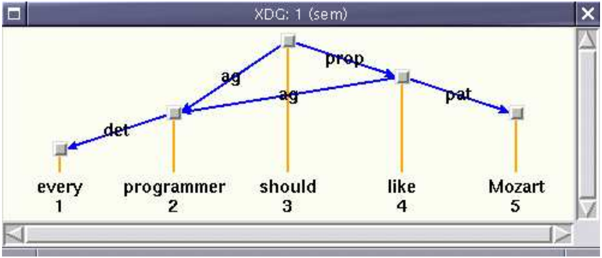

XDG: Main window
Project
Search
Dimensions
Principles
Outputs
Extras
Grammar: MOZ04.ul
Examples: MOZ04.txt
Inspect lexical entries
every programmer should like Mozart
Mozart should like every programmer
Parse
every programmer should like Mozart

Oz Explorer


Oz Explorer
Explorer
Move
Search
Nodes
Hide
Options
Time: 210ms
8
9
0
Depth: 5

Dag display


XDG: 1 (syn)
subj
vcomp
det
obj
every
1
programmer
2
should
3
like
4
Mozart
5


XDG: 1 (sem)
prop
ag
ag
pat
det
every
1
programmer
2
should
3
like
4
Mozart
5

Solver

- based on axiomatization of dependency parsing in (Duchier 1999, 2003)
- factorized into modular, extensible principle library
- principles: sets of constraint functors
- e.g. valency principle: in constraint and out constraint
- starting sequence regulated by global constraint priorities to increase efficiency

Preferences and Search

- idea: guide the search for solutions by external knowledge sources: oracles
- idea by Thorsten Brants and Denys Duchier, extended in (Dienes et al. 2003)
- oracles interact with solver using sockets
- XDK: supports new standard oracle architecture created by Marco Kuhlmann

Overview

Summary

- introduced XDG Development Kit (XDK)
- new lexicon specification language
- grammar file compiler
- graphical interfaces
- solver for parsing and generation
- extensive documentation (200+ pages), PDF, PS, HTML, GNU info

Future Work

- solver: fairly efficient for handcrafted grammars, but not for automatically generated ones
- why? grammar encoding or solver or both?
- theoretical investigation of fragments of XDG
- integration of the new faster GECODE constraint library (Christian Schulte, Guido Tack, Gabor Szokoli)
- super-tagging (lexicon disambiguation before parsing/generation)

References

- Joan Bresnan.
  Lexical Functional Syntax.
  Blackwell, 2001.
- Thorsten Brunklaus.
  Der Oz Inspector — Browsen: Interaktiver, einfacher, effizienter.
  Diploma thesis, Saarland University, 2000.
- Ann Copestake and Dan Flickinger.
  An Open-Source Grammar Development Environment and Broad-Coverage English Grammar Using HPSG.
  In *Conference on Language Resources and Evaluation*, Athens/GRE, 2000.

References

- Benoit Crabbé and Denys Duchier.
  Metagrammar Redux.
  In *Proceedings of the International Workshop on Constraint Solving and Language Processing*, Roskilde/DK, 2004.

- Ralph Debusmann, Denys Duchier, Alexander Koller, Marco Kuhlmann, Gert Smolka, and Stefan Thater.
  A Relational Syntax-Semantics Interface Based on Dependency Grammar.
  In *Proceedings of COLING 2004*, Geneva/SUI, 2004.

- Ralph Debusmann, Oana Postolache, and Maarika Traat.
  A Modular Account of Information Structure in Extensible Dependency Grammar.
  In *Proceedings of the CICLING 2005 Conference*, Mexico City/MEX, 2005. Springer.

References

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

Denys Duchier.
Axiomatizing Dependency Parsing Using Set Constraints.
In *Proceedings of MOL 6*, Orlando/USA, 1999.

Denys Duchier.
Configuration of Labeled Trees under Lexicalized Constraints and Principles.
Research on Language and Computation, 1(3–4):307–336, 2003.

References

- Denys Duchier and Ralph Debusmann.
  Topological Dependency Trees: A Constraint-Based Account of Linear Precedence.
  In *Proceedings of ACL 2001*, Toulouse/FRA, 2001.
- Aravind K. Joshi.
  An Introduction to Tree-Adjoining Grammars.
  In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam/NL, 1987.
- Ronald M. Kaplan and John T. Maxwell.
  LFG Grammar Writer's Workbench.
  Technical report, Xerox PARC, 1996.

References

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

Christian Schulte.
Oz Explorer: A Visual Constraint Programming Tool.
In Lee Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 286–300, Leuven/BEL, jul 1997. MIT Press.

Gert Smolka and Hans Uszkoreit.
NEGRA Project of the Collaborative Research Centre (SFB) 378, 1996–2001.
Saarland University.

References

- Mark Steedman.
  The Syntactic Process.
  MIT Press, Cambridge/USA, 2000.

- Guido Tack.
  Linearisation, Minimisation and Transformation of Data Graphs with Transients.
  Diploma thesis, Saarland University, 2003.

- Michael White.
  Reining in CCG Chart Realization.
  In *Proceedings of the 3rd International Conference on Natural Language Generation*, 2004.

References

XTAG Research Group.
A Lexicalized Tree Adjoining Grammar for English.
Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

Thank you!