Dependency Grammar as Multigraph Description

Ralph Debusmann

Programming Systems Lab, Saarbrücken

IGK Colloquium, December 1st, 2005

Overview

Overview

Two Trends

Two trends in computational linguistics

1. dependency grammar
2. multi-layered linguistic description

Dependency Grammar

- collection of ideas, often attributed to (Tesniere 1959)
  - 1:1-mapping words:nodes
  - head-dependent asymmetry
  - lexicalization
  - valency
- grammar formalisms: have already incorporated most of the ideas (e.g. CCG, HPSG, LFG, TAG)
- statistical parsing: often crucially relies on some of the ideas
- treebanks: some already dependency-based (PDT, Danish DTB), some even being converted (TiGer TB $\rightarrow$ TiGer DTB)

Multi-layered Linguistic Description

- modular approach to linguistics
- largely independent of syntax: research on e.g.
    - predicate-argument structure
    - quantifier scope
    - prosodic structure
    - information structure
    - discourse structure
- treebanks: additional layers, e.g.
    - PDT: predicate-argument structure, information structure
    - TiGer: predicate-argument structure (SALSA) (Erk et al. 2003)
    - Penn TB: predicate-argument structure (PropBank), discourse structure (Penn DTB)

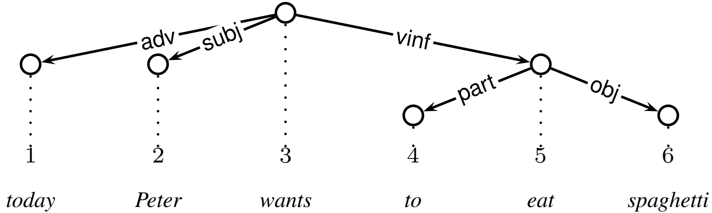
Multi-layered Dependency Grammar

- brings the two trends together
- result: Extensible Dependency Grammar (XDG) (Debusmann et al. 2004 COLING), XDG Development Kit (XDK) (Debusmann et al. 2004 Mozart)
- main problems:
  1. no complete formalization
  2. no efficient large-scale parsing
- this talk: first complete formalization of XDG
- how: as a description language for multigraphs, based on simply typed lambda calculus (Church40)

Overview

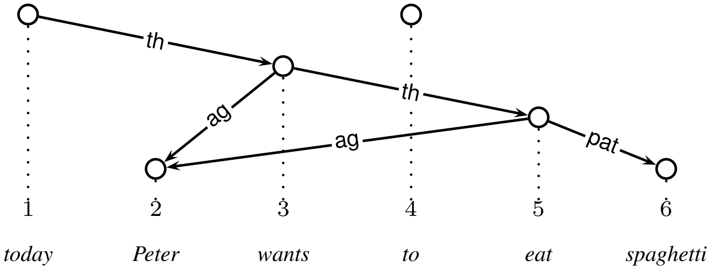
Dependency Graphs

- example dependency graph:



Dependency Graphs (2)

- not restricted to syntactic structure alone:

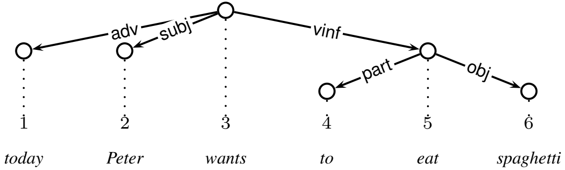


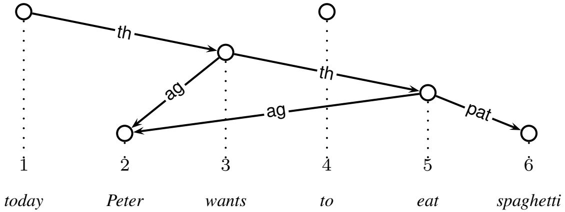
Multigraphs

- multi-layered/multi-dimensional dependency graph:
- consists of an arbitrary number of dependency graphs called dimensions
- all dimensions share the same set of nodes

Multigraphs (2)

- example two-dimensional multigraph (syntax and semantics):





Formalization

- a multigraph is a tuple $(V, Dim, Word, W, Lab, E, Attr, A)$

Components

1. finite set V of nodes (finite interval of the natural numbers starting with 1, therefore totally ordered)
2. finite set $Word$ of words
3. the node-word mapping $W \in V \to Word$
4. a finite set Lab of edge labels
5. a set $E \subseteq V \times V \times Dim \times Lab$ of labeled directed edges
6. a finite set $Attr$ of attributes
7. the node-attributes mapping $A \in V \to Dim \to Attr$

Relations

- each dimension $d \in Dim$ associated with four relations:

Four Relations

1. labeled edge: $\stackrel{\cdot}{\longrightarrow}_d$
2. edge: $\rightarrow_d$
3. dominance (strict): $\rightarrow_d^+$
4. precedence: $v \prec v'$

Overview

Types

- $T \in Ty$ given a set At of atoms (arbitrary symbols):

Types

$$
\begin{array}{rcll}
T & ::= & \mathsf{B} & \text{boolean} \\
 & \mid & \mathsf{V} & \text{node} \\
 & \mid & T_1 \to T_2 & \text{function} \\
 & \mid & \{a_1, \ldots, a_n\} & \text{finite domain} \\
 & \mid & \{a_1 : T_1, \ldots, a_n : T_n\} & \text{record}
\end{array}
$$

Interpretation

- B as $\{0,1\}$
- V as (finite interval of the natural numbers starting with 1)
- $T_1 \rightarrow T_2$ 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 by their: dimensions, words, labels, attributes
- multigraph type: tuple $M = (dim, word, lab, attr)$, where:

Multigraph Type

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, i.e. the finite domain of the edge labels on that dimension
4. $attr \in dim \to Ty$ is a function from dimensions to attributes types, i.e. the (arbitrary) type of the attributes on that dimension

Multigraphs and Multigraph Types

- $\mathcal{M}\ T$: 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

Conditions

1. The dimensions are the same
2. The words are the same
3. The edges in E have the right edge labels for their dimension according to lab
4. The nodes have the right attributes for their dimension according to $attr$

Terms

- defined given set At of atoms and Con of constants:

Terms

$t ::=$	x	variable
$\mid$	c	constant
$\mid$	$\lambda x : T.t$	abstraction
$\mid$	$t_1\ t_2$	application
$\mid$	a	atom
$\mid$	$\{a_1 = t_1, \ldots, a_n = t_n\}$	record
$\mid$	$t.a$	record selection

Signature

- determined by a multigraph type $M = (dim, word, lab, attr)$
- two parts:
  1. the logical constants
  2. the multigraph constants

Logical Constants

- include the type constant B and the following term constants:

Logical Constants

0	:	B	false
1	:	B	true
$\neg$	:	$\mathsf{B} \to \mathsf{B}$	negation
$\vee, \wedge, \Rightarrow, \Leftrightarrow$	:	$\mathsf{B} \to \mathsf{B} \to \mathsf{B}$	disjunction,conjunction etc.
$\doteq_T, \dot{\neq}$	:	$T \to T \to \mathsf{B}$	equality, inequality
$\exists_T, \exists^1_T, \forall_T$	:	$(T \to \mathsf{B}) \to \mathsf{B}$	quantification

Multigraph Constants

- include the type constant V and the following term constants:

Multigraph Constants

$\xrightarrow{\cdot}_d$	:	$\mathsf{V} \to \mathsf{V} \to \mathit{lab}\ d \to \mathsf{B}$	labeled edge
$\to_d$	:	$\mathsf{V} \to \mathsf{V} \to \mathsf{B}$	edge
$\to_d^+$	:	$\mathsf{V} \to \mathsf{V} \to \mathsf{B}$	dominance
$\prec$	:	$\mathsf{V} \to \mathsf{V} \to \mathsf{B}$	precedence
$(\mathit{word}\ \cdot)$	:	$\mathsf{V} \to \mathit{word}$	word
$(d\ \cdot)$	:	$\mathsf{V} \to \mathit{attr}\ d$	attributes

Multigraph Constants (2)

- where we interpret:

Interpretation

- $\stackrel{\cdot}{\longrightarrow}_d$ as the labeled edge relation on dimension d.
- $\rightarrow_d$ 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:
  1. a multigraph type M
  2. 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 that:
  1. have multigraph type M
  2. satisfy all principles P

String Language

- given a grammar $G = (M, P)$, the string language $L(G)$ is the set of strings $s = w_1 \dots w_n$ such that:
  1. there is a model of G with equally many nodes as words:

     $$V = \{1, \dots, n\}$$

  2. the concatenation of the words of the nodes of this model yields s:

     $$(\mathit{word}\ 1) \dots (\mathit{word}\ n) = s$$

Recognition Problem

- two kinds (Trautwein 1995):

Recognition Problem

1. universal recognition problem: given a pair (G, s) where G is a grammar and s a string, is s in the language generated by G?
2. fixed recognition problem: let there be a fixed grammar G. Given a string s, is s in the language generated by G?

Complexity of the Recognition Problems

- fixed recognition problem: NP-hard
- proof: reduction of the SAT problem
- universal recognition problem: also NP-hard
- proof: implied by the above proof, individually: by reduction of the Hamiltonian Path problem, inspired by (Koller and Striegnitz 2002)
- no upper bound proven yet, with this formalization: probably worse than NP for both, with reasonable restrictions on the principles, conjecture: in NP

Parsing

- constraint-based parser in the XDK implementation already efficient for handcrafted grammars despite the intractable complexity
- but not suitable for large-scale parsing
- finding polynomial fragments of XDG and improving the efficiency of the parser: after my thesis

Overview

Dependency Grammar

- collection of ideas, often attributed to (Tesniere 1959)
  - 1:1-mapping words:nodes
  - head-dependent asymmetry
  - lexicalization
  - valency
- in addition for XDG:
  - order
  - projectivity
  - multi-dimensionality

1:1-mapping words:nodes

- node-word mapping
- recall: multigraphs are tuples $(V, \mathit{Dim}, \mathit{Word}, W, \mathit{Lab}, E, \mathit{Attr}, A)$

Components

1. $\ldots$
2. the node-word mapping $W \in V \to \mathit{Word}$
3. $\ldots$

Head-Dependent Asymmetry

- labeled directed edges
- again recall: multigraphs are tuples $(V, Dim, Word, W, Lab, E, Attr, A)$

Components

1. ...
2. a set $E \subseteq V \times V \times Dim \times Lab$ of labeled directed edges
3. ...

Lexicalization

- idea: behavior of the nodes depends on the associated words
- to model lexicalization in XDG, we split the attributes into:
    1. lexical attributes
    2. non-lexical attributes

Attributes

- formally, the attributes $attr\ d$ of each dimension d must be a record of the type:

$$attr\ d = \left\{ \begin{array}{ccc} lex & : & L \\ a_1 & : & \ldots \\ & \ldots & \\ a_n & : & \ldots \end{array} \right\}$$

- where lex harbors the lexical attributes, which have type L, and $a_1, \ldots a_n$ the non-lexical attributes

Lexicon

- the lexicon is a set of lexical entries of type E:

$$E = \left\{ \begin{array}{rcl} word & : & word \\ d_1 & : & L_1 \\ & \dots & \\ d_m & : & L_m \end{array} \right\}$$

- where each lexical entry is associated with a word by feature $word$, and specifies the lexical attributes of the dimensions $d_1, \dots, d_m$

Lexicalization Principle

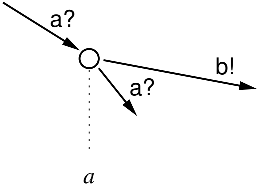
- realizes lexicalization:
  1. for each node, a lexical entry e must be selected from the lexicon $lexicon$
  2. e must be associated with same word with which the node is associated.
  3. e determines the lexical attributes

Lexicalization in XDG

$$
\begin{array}{ll}
1. & \exists e \in lexicon \quad \wedge \\
2. & e.word \doteq (word\ v) \quad \wedge \\
3. & (d_1\ v).lex \doteq e.d_1 \quad \wedge \\
 & \ldots \\
 & (d_m\ v).lex \doteq e.d_m
\end{array}
$$

Valency

- idea: lexically specify for each node its licensed incoming and outgoing edges
- leads to notion of configuration of fragments which need to be assembled to yield analyses
- example fragment:

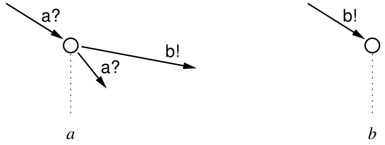


Example Grammar

- example grammar:

$$L_1 \quad = \quad \{w \in (a \cup b)^+ \mid |w|_a = |w|_b\}$$

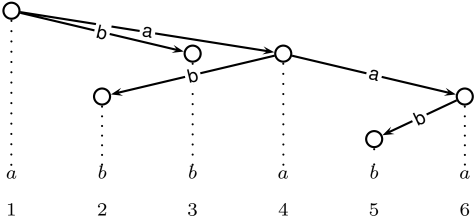
- idea: use the following fragments:



- models must be trees

Example Analysis

- example analysis:



- intuitively: as arranged in a chain, each a must have one outgoing edge to a b
- this ensures that there are equally many as and bs

Valency in XDG

- no incoming edges labeled l for node v on dimension d:

$$\mathit{in0}\,\langle d \rangle\ v\ l \quad = \quad \neg\exists v' : v' \xrightarrow{l}_d v$$

- precisely one incoming edge labeled l:

$$\mathit{in1}\,\langle d \rangle\ v\ l \quad = \quad \exists^1 v' : v' \xrightarrow{l}_d v$$

- at most one incoming edge labeled l:

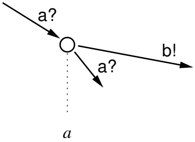
$$\mathit{in0or1}\,\langle d \rangle\ v\ l \quad = \quad (\mathit{in0}\langle d \rangle\ v\ l) \vee (\mathit{in1}\,\langle d \rangle\ v\ l)$$

Lexicalization of Valency

- idea: express fragments by lexical entries
- two lexical attributes in and out
- map edge labels to cardinalities:
  - ! (precisely one edge)
  - ? (at most one edge)
  - $*$ (arbitrary many edges)
  - 0 (no edges)

Lexicalization of Valency (2)

- example fragment:



- corresponding lexical entry:

$$\left\{ \begin{array}{rcl} word & = & a \\ \mathsf{ID} & = & \left\{ \begin{array}{rcl} in & = & \{a = ?, b = 0\} \\ out & = & \{a = ?, b = !\} \end{array} \right\} \end{array} \right\}$$

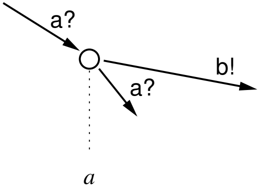
Valency Principle

- realizes lexicalized valency:

$$
\begin{array}{l}
valency\langle d\rangle = \\
\forall l: \\
\quad \begin{array}{lcccl}
(d\ v).lex.in.l & \doteq & 0 & \Rightarrow & in0\ v\ l \\
(d\ v).lex.in.l & \doteq & ! & \Rightarrow & in1\ v\ l \\
(d\ v).lex.in.l & \doteq & ? & \Rightarrow & in0or1\ v\ l \\
 & & & & \\
(d\ v).lex.in.l & \doteq & 0 & \Rightarrow & out0\ v\ l \\
(d\ v).lex.in.l & \doteq & ! & \Rightarrow & out1\ v\ l \\
(d\ v).lex.in.l & \doteq & ? & \Rightarrow & out0or1\ v\ l
\end{array}
\end{array}
$$

Order

- idea: fragments so far unordered, now we make them ordered
- lexical order on the dependents of each node, in addition, the mother is ordered with respect to its dependents:
- additional edge label m ("mother"): position of the mother with respect to its dependents



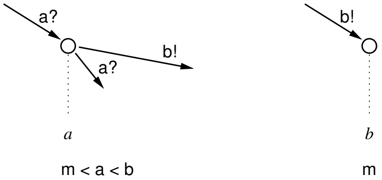
m < a < b

Example Grammar

- example grammar:

$$L_2 = \{w \in a^n b^n \mid n \geq 1\}$$

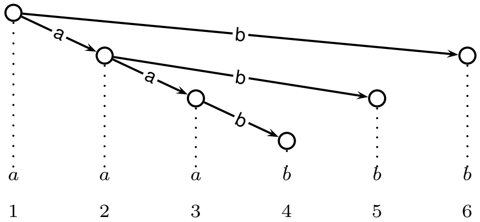
- idea: use the following ordered fragments:



- models must be trees

Example Analysis

- example analysis:



- like before: as arranged in a chain, and each a must have one outgoing edge to a b (equally many as and bs)
- in addition: all as precede all bs, and all mothers precede their dependents

Order in XDG

- make the daughters with incoming edge label a precede those with incoming edge label b:

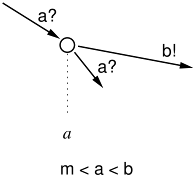
$$v \xrightarrow{a}_d v' \wedge v \xrightarrow{b}_d v'' \quad \Rightarrow \quad v' \prec v''$$

- make all mothers precede their daughters:

$$v \rightarrow_d v' \quad \Rightarrow \quad v \prec v'$$

Lexicalization of Order

- lexical attribute $order$, type: set of pairs of edge labels (including m) representing a strict partial order
- e.g. ordered fragment:



- corresponding lexical entry:

$$\left\{ \begin{array}{rcl} word & = & a \\ d & = & \left\{ \begin{array}{rcl} in & = & \{a = ?, b = 0\} \\ out & = & \{a = ?, b = !\} \\ order & = & \{(m,a),(m,b),(a,b)\} \end{array} \right\} \end{array} \right\}$$

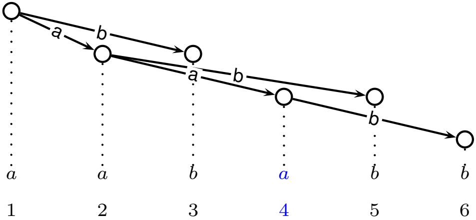
Order Principle

- realizes lexicalized order:

$$
\begin{array}{l}
order\langle d\rangle = \\
\forall (l, l') \in (d\ v).lex.order : \\
\begin{array}{ccccccc}
v \xrightarrow{l}_d v' & \wedge & v \xrightarrow{l'}_d v'' & \Rightarrow & v' \prec v'' & \wedge \\
l \doteq m & \wedge & v \xrightarrow{l'}_d v' & \Rightarrow & v \prec v' & \wedge \\
v \xrightarrow{l}_d v' & \wedge & l' \doteq m & \Rightarrow & v' \prec v &
\end{array}
\end{array}
$$

Projectivity

- problem: locally ordering the daughters does not suffice to model L_2
- counter example: all a-daughters precede the b-daughters, and all mothers precede their daughters, yet not all as precede all bs:



Projectivity (2)

- problem: we need to ensure that we do not only order the daughters but entire subtrees
- idea: forbid edges to cross projection edges of nodes higher up in the graph, i.e. enforce projectivity

Projectivity Principle

- realizes projectivity:

$$
\begin{array}{l}
projectivity\langle d \rangle = \\
\quad v \longrightarrow_d v' \wedge v \prec v' \;\Rightarrow\; \forall v'' : v \prec v'' \wedge v'' \prec v' \;\Rightarrow\; v \longrightarrow_d^+ v'' \wedge \\
\\
\quad v \longrightarrow_d v' \wedge v' \prec v \;\Rightarrow\; \forall v'' : v' \prec v'' \wedge v'' \prec v \;\Rightarrow\; v \longrightarrow_d^+ v''
\end{array}
$$

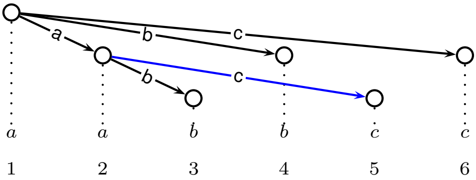
Multi-dimensionality

- additional expressivity and modularity, e.g. to model other layers of linguistic description
- example here: two dimensions to model the non-context-free language L_3:

$$L_3 = \{w \in a^n b^n c^n \mid n \geq 1\}$$

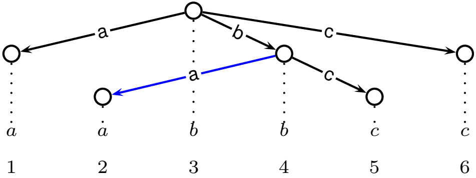
One Dimension is Not Enough

- impossible to find one-dimensional analyses for blocks > 1 which are projective
- if the root is an a or a c, there is no way to connect as and cs of depth > 1 without crossing the projection edges of the bs higher up:



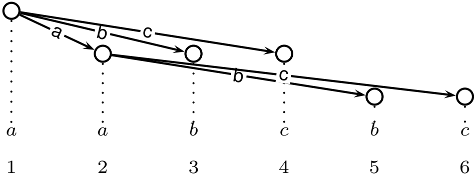
One Dimension is Not Enough (2)

- if the root is a b, there is no way to connect the bs with depth > 1 to both the corresponding as and cs without crossing projection edges of the bs higher up:



One Dimension is Not Enough (3)

- cannot drop projectivity, because this would inevitably lead to overgeneration, e.g.:

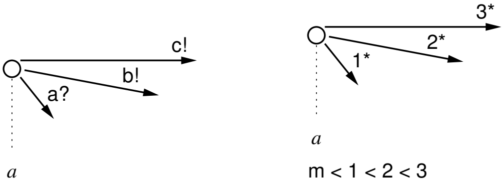


Multi-dimensionality to the Rescue

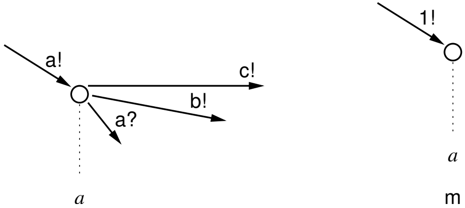
- idea: disentangle counting and ordering using two dimensions:
  1. the Immediate Dominance (ID) dimension for counting, i.e. to ensure that for each a, there is precisely one b and one c
  2. the Linear Precedence (LP) dimension for ordering, i.e. to ensure that all as precede all bs which precede all cs
- ID dimension: unordered tree
- LP dimension: ordered tree

Grammar

- a as the root (left: ID, right: LP):

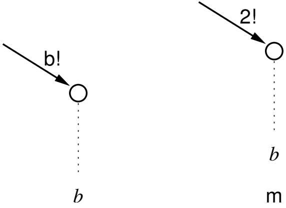


- a as a dependent:

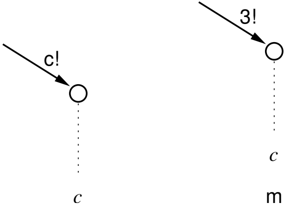


Grammar (2)

- b:

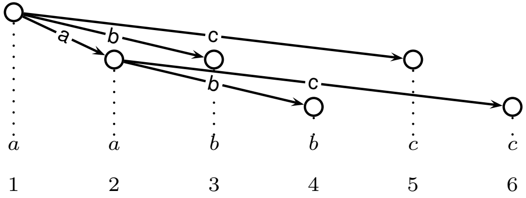


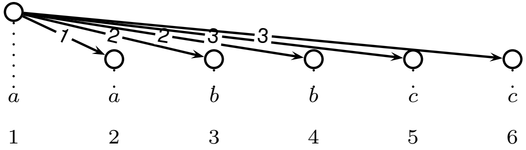
- c:



Example Analysis

- top: ID, bottom: LP:





Possible Extensions

- interesting: grammar can be straightforwardly extended to handle languages like e.g. $a^n b^n c^n d^n$ and $a^n b^n c^n d^n e^n$
- cannot be handled by e.g. TAG (Joshi 1987) and CCG (Steedman 2000) (mildly context-sensitive)

Overview

Conclusions

- XDG: very expressive and modular grammar formalism, brings together two recent trends in computational linguistics
- presented first complete formalization
- showed how to realize the ideas of dependency grammar
- basis for future work on multi-dimensional dependency grammar, in particular:
  - finding fragments with tractable complexity
  - developing more efficient parsers

Thanks for your attention!

References

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

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/CH, 2004.

Ralph Debusmann, Denys Duchier, and Joachim Niehren.
The XDG Grammar Development Kit.
In *Proceedings of the MOZ04 Conference*, volume 3389 of *Lecture Notes in Computer Science*, pages 190–201, Charleroi/BE, 2004. Springer.

References

Katrin Erk, Andrea Kowalski, Sebastian Pado, and Manfred Pinkal.
Towards a Resource for Lexical Semantics: A Large German Corpus with Extensive Semantic Annotation.
In *Proceedings of ACL 2003*, Sapporo/JP, 2003.

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.

Alexander Koller and Kristina Striegnitz.
Generation as Dependency Parsing.
In *Proceedings of ACL 2002*, Philadelphia/US, 2002.

References

- Mark Steedman.
  The Syntactic Process.
  MIT Press, Cambridge/US, 2000.
- Lucien Tesnière.
  Eléments de Syntaxe Structurale.
  Klincksiek, Paris/FR, 1959.