

Hello. Welcome to my presentation of the paper “Multi-dimensional Dependency Grammar as Graph Description”, which was written by Gert Smolka and myself, Ralph Debusmann from the Programming Systems Lab in Saarbruecken, Germany

# Multi-dimensional Dependency Grammar as Graph Description

Ralph Debusmann and Gert Smolka

Programming Systems Lab, Saarbrücken, Germany

FLAIRS-19, May 11th, 2006

In this presentation, I will present the first complete formalization of the grammar formalism of Extensible Dependency Grammar, and prove the lower bound of its computational complexity. We start out with two trends in natural language processing...

# Overview

- 1 Introduction
- 2 Extensible Dependency Grammar—the First Formalization
- 3 Computational Complexity
- 4 Conclusions

# Overview

- 1 Introduction
- 2 Extensible Dependency Grammar—the First Formalization
- 3 Computational Complexity
- 4 Conclusions

...namely, dependency grammar and multi-layered linguistic description

# Two Trends in Natural Language Processing

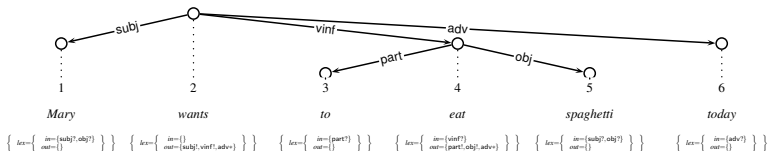
- dependency grammar (Tesnière 1959), (Mel'čuk 1988)
- multi-layered linguistic description

What is dependency grammar? It's a collection of ideas for the syntactic analysis of natural language, which can best be explained together with a dependency analysis such as the one below for the sentence "Mary wants to eat spaghetti today". Dependency analyses are graphs where each word is mapped 1:1 to a node (point), and the edges are labeled by dependency relations like subject (point) and adverb (point), i.e., "Mary" is the subject, and "today" an adverb of "wants". The nodes are associated with lexical information in records, which e.g. determine the valency of the nodes, i.e., their licensed incoming and outgoing edges. For example, the node corresponding to "wants" licenses no incoming edges, and requires one outgoing edge labeled subj, one labeled vinf (for "verbal complement in infinitival form"), and licenses arbitrary outgoing edges labeled "adv".



# Dependency Grammar

- collection of ideas for the analysis of natural language
- example analysis of *Mary wants to eat spaghetti today*:



- graph, 1:1-mapping nodes:words, dependency relations, valency
- e.g.: *wants*:

$$\left\{ lex = \left\{ \begin{matrix} in = \{ \} \\ out = \{subj!, vinf!, adv*\} \end{matrix} \right\} \right\}$$

These ideas from dependency grammar are becoming more and more important. They have been incorporated more and more in modern grammar formalisms. For statistical parsing, dependency relations are indispensable, and more and more treebanks directly based on dependency grammar are coming up.

# Dependency Grammar as a trend

- incorporated into grammar formalisms: CCG (Steedman 2000), HPSG (Pollard/Sag 1994), LFG (Bresnan/Kaplan 1982), TAG (Joshi 1987)
- indispensable for statistical parsing (Collins 1999)
- treebanks: Prague Dependency Treebank (Bohmová et al. 2001), Danish Dependency Bank, TiGer Dependency Bank (Forst et al. 2004)

Another trend, especially with respect to treebanks, is the addition of new layers of linguistic description, e.g. predicate-argument structure, information structure or even discourse structure. Interestingly, these new layers are often dependency-based. Thus, a straightforward question is whether we could represent these layers as modules in \*one\* framework, ideally based on dependency grammar.

# Multi-layered Linguistic Description

- additional layers of annotation
- predicate-argument structure: PropBank (Kingsbury/Palmer 2002), SALSA (Erk et al. 2003), tectogrammatical structure of the PDT
- information structure: PDT
- discourse structure: Penn Discourse Treebank (Webber et al. 2005)
- annotation: mostly dependency-based
- can we represent these layers as modules in one framework based on dependency grammar?

Extensible Dependency Grammar (XDG) is a new dependency-based framework which is perfectly capable of this. It supports arbitrary many layers of linguistic description called “dimensions”. Each dimension is an individual dependency graph, but crucially, all dimensions share the same set of nodes. XDG is a model-theoretic framework: its models are called “multigraphs”.

# Extensible Dependency Grammar (XDG)

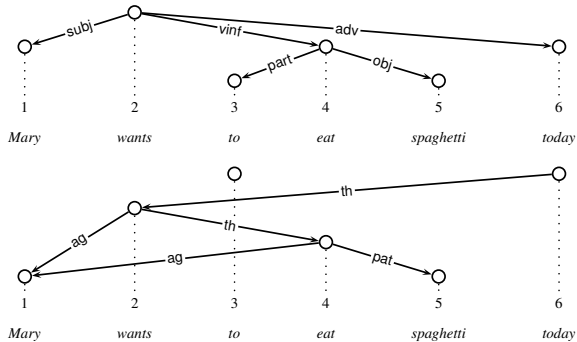
- new grammar formalism (Debusmann 2006 PhD)
- supports arbitrary many layers of linguistic description called “dimensions”, all sharing the same set of nodes
- model-theoretic: models called “multigraphs”

Here is an example XDG multigraph using two dimensions: syntax (as before, point) and predicate-argument structure (point). As you can see, both dimensions share the same set of nodes: e.g. the node corresponding to “Mary” is the subject of “wants” on the syntax dimension, and simultaneously the agent of both “wants” and “eat” on the predicate-argument structure dimension.



# Multigraph

- syntax and predicate-argument structure:



We have implemented a parser for XDG based on constraint programming in the Mozart/Oz programming language, and an extensive grammar development kit around it, the XDK, of which we show a screenshot below.

# Implementation

- concurrent constraint-based parser written in Mozart/Oz
- (Mozart06)
- XDG Development Kit (XDK) (Debusmann et al. 2004 MOZ)

The screenshot displays the XDG Development Kit (XDK) interface, which is used for developing and running the concurrent constraint-based parser. The interface consists of several windows:

- Project:** A window showing project settings, including the project name (XDG Development Kit), version (1.4.2), and author information (Ralph Debusmann).
- Parser Selection:** A window showing the selected parser (Mozart/Oz) and the selected grammar (XDG Development Kit).
- Graphs:** Several windows showing dependency graphs for the sentence "Mary wants to eat spaghetti today". The graphs illustrate the dependencies between words and their corresponding nodes in the dependency grammar.
- Parser Selection Options:** A window showing the options for the parser, including the parser type (Mozart/Oz), the grammar (XDG Development Kit), and the parser options (Mozart/Oz).
- Parser Selection:** A window showing the selected parser (Mozart/Oz) and the selected grammar (XDG Development Kit).
- Parser Selection:** A window showing the selected parser (Mozart/Oz) and the selected grammar (XDG Development Kit).
- Parser Selection:** A window showing the selected parser (Mozart/Oz) and the selected grammar (XDG Development Kit).
- Parser Selection:** A window showing the selected parser (Mozart/Oz) and the selected grammar (XDG Development Kit).

We have applied XDG to linguistics by modeling fragments of German, Arabic and English syntax, we have used it to create the first relational (not functional) syntax-semantics interface, and a modular account of the prosodic account of information structure by Steedman.

# Application

- German syntax (Duchier/Debusmann 2001 ACL), (Debusmann 2001), (Bader et al. 2004)
- Arabic syntax (Odeh 2004)
- English syntax (Debusmann 2006 PhD)
- relational syntax-semantics interface (Debusmann et al. 2004 COLING)
- prosodic account of information structure (Debusmann et al 2005 CICLING)

But: XDG has so far been hindered by two stumbling blocks: 1) it has not yet been completely formalized, and 2) the XDG parser does not yet scale up for large-scale parsing. In this paper, we tackle the first stumbling block: we provide the first complete formalization of XDG as a description language for multigraphs.

## Two Stumbling Blocks

- 1 no complete formalization (Debusmann et al. 2005 FG-MOL)
- 2 no efficient large-scale parsing (Bojar 2004), (Moehl 2004), (Narendranath 2004)

# Overview

- 1 Introduction
- 2 Extensible Dependency Grammar—the First Formalization**
- 3 Computational Complexity
- 4 Conclusions



We formalize XDG as a description language for multigraphs in higher order logic, expressed in simply typed lambda calculus, which we extend with finite domains and records. The types of XDG are the following: there are two finite base types, booleans and nodes, functions, finite domains and records. The interpretation of the boolean base type is the set of zero and one, and the interpretation of the node base type a finite interval of the natural numbers starting with 1. Thus, both base types are finite, and therefore also the models of XDG.

# A Description Language for Multigraphs

- formalization as a description language for multigraphs in higher order logic
- expressed in simply typed lambda calculus extended with finite domains and records
- types, given set of atoms  $At$ :

$a \in At$		
$T \in Ty$	$::=$	$B$ boolean
		$V$ node
		$T_1 \rightarrow T_2$ function
		$\{a_1, \dots, a_n\}$ finite domain ( $n \geq 1$ )
		$\{a_1 : T_1, \dots, a_n : T_n\}$ record

- interpretation:  $B = \{0, 1\}$ ,  $V = \{1, 2, \dots, n\}$  given  $n$  nodes, i.e., both base types finite

The signature of XDG varies according to the dimensions, words, edge labels and attributes of the described multigraphs. We capture this using the notion of a “multigraph type”, where the domains of dimensions and words must be finite.

# Multigraph Type

- signature of XDG varies according to the dimensions, words, edge labels and attributes of the described multigraphs
- multigraph type:  $MT = (Dim, Word, lab, attr)$
- domains of dimensions and words must be finite

The signature of XDG is now defined given a multigraph type, and consists of multigraph constants and equality. The multigraph constants include the function “labeled edge” for two nodes and an edge label on a dimension  $d$  in  $\text{Dim}$ , the function “precedence” for two nodes, the node-word mapping and the node-attributes mapping. In addition, XDG defines the logical constant for equality, with which we can axiomatize all the other logical operators, as usual in HOL.

# Signature

- multigraph constants, given multigraph type

$MT = (Dim, Word, lab, attr):$

$\xrightarrow{d}$	: $V \rightarrow V \rightarrow lab\ d \rightarrow B$	labeled edge ( $d \in Dim$ )
$<$	: $V \rightarrow V \rightarrow B$	precedence
$(W \cdot)$	: $V \rightarrow Word$	node-word mapping
$(d \cdot)$	: $V \rightarrow attr\ d$	node-attributes mapping ( $d \in Dim$ )

- logical constant:

$\doteq_T$  :  $T \rightarrow T \rightarrow B$  equality (for each type  $T$ )

An XDG grammar is now determined by a multigraph type and a set of formulas called principles, which stipulate the well-formedness conditions of the grammar. The models of an XDG grammar are all multigraphs with multigraph type  $MT$  and which satisfy  $P$ , and the string language of an XDG grammar is the set of all strings  $s$  of words  $w_1$  to  $w_n$  such that there are as many nodes as words, and the concatenation of the words of the nodes yields  $s$ .

# Grammar, models and string language

- grammar:  $G = (MT, P)$
- $P$  set of formulas called “principles”, i.e., the well-formedness conditions
- models: all multigraphs with multigraph type  $MT$  and which satisfy  $P$
- string language: set of all strings  $s = w_1 \dots w_n$  such that:
  - 1 there are as many nodes as words:  $V = \{1, \dots, n\}$
  - 2 concatenating the words of the nodes yields  $s$ :  
 $(W\ 1) \dots (W\ n) = s$



Now what precisely are the principles of XDG? Here is an example, the tree principle. Given a dimension  $d$  (point), it states that the graph on  $d$  must have no cycles (point), that there is only one root (point), and that each node has at most one incoming edge (point).

# Tree Principle

- three conditions:
  - ① There are no cycles.
  - ② There is precisely one root.
  - ③ Each node has at most one incoming edge.
- principle definition:

$$\begin{aligned}
 tree_d = & \forall v : \neg(v \rightarrow_d^+ v) \quad \wedge \\
 & \exists^1 v : \neg \exists v' : v' \rightarrow_d v \quad \wedge \\
 & \forall v : (\neg \exists v' : v' \rightarrow_d v) \vee (\exists^1 v' : v' \rightarrow_d v)
 \end{aligned}$$

For its application to natural language, we have defined a number of additional principles, including principles stipulating DAGness, valency, order, projectivity, agreement, linking etc.

## Other Principles

- DAG
- valency
- order
- projectivity
- agreement
- linking
- etc. (Debusmann 2006 PhD)

# Overview

- 1 Introduction
- 2 Extensible Dependency Grammar—the First Formalization
- 3 Computational Complexity**
- 4 Conclusions

That's XDG. Now, we turn to its computational complexity, more precisely, of its recognition problems. The more general universal recognition problem is defined as follows: given a pair  $G,s$  of a grammar and a string, is  $s$  in the language of  $G$ ? The fixed recognition problem is defined such: let  $G$  be a \*fixed\* grammar, given a string  $s$ , is  $s$  in the language of  $G$ ? Our plan is to prove that the fixed recognition problem of XDG is NP-hard, from which NP-hardness of the universal recognition problem then falls out.

# Recognition Problems

- universal recognition problem: given a pair  $(G, s)$  where  $G$  is a grammar and  $s$  a string, is  $s$  in  $L(G)$ ?
- fixed recognition problem: let  $G$  be a fixed grammar. Given a string  $s$ , is  $s$  in  $L(G)$ ?
- plan: prove NP-hardness of the fixed recognition problem, NP-hardness of the universal then falls out

To prove that the fixed recognition problem is NP-hard, we use a reduction the NP-complete SAT problem to the XDG recognition problem. The SAT problem poses the question whether a propositional formula has an assignment that evaluates to true. For our purposes, we define propositional formulas as either variables (point), false (point), or an implication of two formulas (point).



# Reduction

- proof by reducing the NP-complete SAT problem to the fixed XDG recognition problem
- SAT: does a propositional formula  $f$  have an assignment that evaluates to true?
- propositional formula:

$f ::=$	$X, Y, Z, \dots$	variable
	$0$	false
	$f_1 \Rightarrow f_2$	implication

In preparing the input to the XDG recognition problem, we face two problems: 1) propositional formulas as defined above can be ambiguous. 2) propositional formulas can contain arbitrary many variables, but our fixed grammar can only have a finite set of words. The input preparation function `prep` solves the first problem by transforming the formula into unambiguous prefix notation, and the second by adopting a unary encoding of variables, where the leftmost one corresponds to “var 1”, the next to “var 1 1” etc.

# Input Preparation

- 2 challenges:
  - 1 propositional formulas can be ambiguous
  - 2 can contain arbitrary many variables, but an XDG grammar only has a finite set of words
- input preparation function:  $prep : f \rightarrow Word$
- example formula:  $(X \Rightarrow Y) \Rightarrow Y$

- 1 prefix notation:

$$\Rightarrow \Rightarrow X Y Y$$

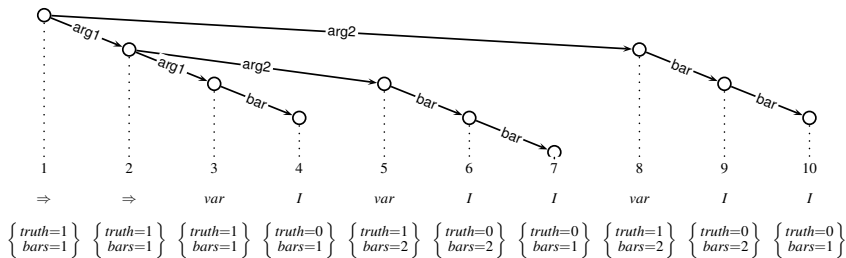
- 2 unary encoding:

$$\Rightarrow \Rightarrow var \mid var \mid \mid var \mid \mid$$

We model the formulas in XDG as ordered trees representing their structure, where each node corresponding to an implication has one outgoing edge to its antecedent (arg1), and one to its consequent (arg2, point). Variables have one outgoing edge to the first bar to their right (point), and the same for bars (point). The nodes have two attributes: “truth” of type B representing the truth value of the node, and “bars”, counting the number of bars to the right of the variables. We use the latter attribute to ensure coreference of variable occurrences: all variable nodes having the same bars value are coreferent, i.e, must have the same truth value. E.g., the truth value of node 5 is 1 and it has two bars to its right, i.e., its bars value is 2. Node 8 has the same bars value and is thus coreferent with node 5 (point). Node 3 has only 1 bar to its right and is thus not coreferent with the others.

## Models

- representation of the example formula  $(X \Rightarrow Y) \Rightarrow Y$ :

$$\Rightarrow \Rightarrow \text{var} \mid \text{var} \mid \mid \text{var} \mid \mid$$


Now, the question remains what type we should choose for the “bars” attribute. Recall that XDG only has the finite base types B (boolean) and V (nodes). We can only choose V, whose interpretation is a finite interval of the natural numbers starting with 1: firstly, there are always more nodes in the analysis than variables in the formula, i.e., V includes enough elements to distinguish the variables, and secondly, we can count the bars by emulating incrementation with the precedence predicate:  $v$  is the result of incrementing  $v'$  if  $v$  is bigger than  $v'$  and there is no node  $v''$  between the two.

# Coreference

- which type for the “bars” attribute?
- idea: use  $V$ , whose interpretation is a finite interval of the natural numbers starting with 1, because:
  - 1 there are always more nodes in the analysis than variables in the formula, i.e.,  $V$  always includes enough elements to distinguish all variables
  - 2 bars can be counted by emulating incrementation with the precedence predicate:

$$incr = \lambda v, v'. v < v' \wedge \neg \exists v'' : v < v'' \wedge v'' < v'$$

The proof that the lower bound of the computational complexity of the fixed recognition problem for XDG is NP-hard now goes as follows: given a formula  $f$  and the fixed XDG grammar  $G$  defined above,  $f$  is satisfiable if and only if prep of  $f$  is in the language of  $G$ , i.e., SAT is reducible to the fixed recognition problem for XDG. As the reduction is polynomial, the fixed recognition problem for XDG is NP-hard. As the fixed recognition problem is an instance of the more general universal recognition problem, the universal recognition problem is also NP-hard.



## NP-hardness of the Fixed Recognition Problem

- Given a formula  $f$  and the fixed XDG grammar  $G$  defined above,  $f$  is satisfiable if and only if  $prep f \in L(G)$ , i.e., SAT is reducible to the fixed recognition problem for XDG.
- as the reduction is polynomial, the fixed recognition problem for XDG is NP-hard
- universal recognition problem: generalization of the fixed recognition problem, thus also NP-hard

Different restrictions of the principles, lead to different upper bounds of the computational complexity. If we restrict the principles to be first order, the upper bound is in PSPACE. If we further restrict the principles to be testable in polynomial time, which is true for all the principles developed so far, the upper bound is in NP.

# Upper Bounds

- principles first order: upper bound in PSPACE
- principles testable in polynomial time: upper bound in NP (all principles defined so far)

# Overview

- 1 Introduction
- 2 Extensible Dependency Grammar—the First Formalization
- 3 Computational Complexity
- 4 Conclusions**

XDG is a showcase for two trends in NLP: dependency grammar and multi-layered linguistic description. But, so far, there were two tumbling blocks: no complete formalization, no efficient large-scale parsing. In This talk, we have tackled the first stumbling block by giving the first complete formalization of XDG as a description language for multigraphs. The complexity of the XDG recognition problem is NP-hard, and the upper bound of the complexity depends on the restrictions imposed on the principles. For realistic grammars, the upper bound seems to be in NP (not proven).

# Summary

- XDG is a showcase for two trends in NLP: dependency grammar and multi-layered linguistic description
- but: two stumbling blocks: no complete formalization, no efficient large-scale parsing
- this talk: first complete formalization of XDG as a description language for multigraphs
- complexity: NP-hard, upper bound: with realistic restrictions: in NP

The existing constraint-based parser is complete, i.e., includes implementations for all principles devised in the context of XDG, is concurrent, i.e., able to process all dimensions of linguistic description simultaneously, and is also relatively efficient on the handcrafted grammars developed so far. However, the parser does not yet scale up to large-scale parsing. We plan to attack this problem from two directions: the first consists of optimizing the existing constraint-based parser, e.g. by finding global constraints, using the new, faster Gecode constraint library, and by adding statistical support. The second consists of finding polynomially parsable fragments of XDG, e.g. fragments related to TAG, STAG or GMTG, for which polynomial parsers do exist.

# Future Work

- XDG parser: constraint-based parser, complete, concurrent, efficient for handcrafted grammars
- but does not yet scale up to large-scale parsing
- future work:
  - 1 optimizing the constraint-based parser: find global constraints, Gecode (Schulte/Stuckey 2004), (Schulte/Tack 2005), statistical support (supertagging)
  - 2 finding polynomially parsable fragments of XDG, e.g. related to TAG, STAG or GMTG (Melamed et al. 2004)



That's it, thanks a lot for your attention!

Thanks for your attention!

# References

-  Regine Bader, Christine Foeldes, Ulrich Pfeiffer, and Jochen Steigner.

Modellierung grammatischer Phänomene der deutschen Sprache mit Topologischer Dependenzgrammatik, 2004. Softwareprojekt, Saarland University.

-  Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká.

The Prague Dependency Treebank: Three-level annotation scenario.

In *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers, 2001.

# References



Ondrej Bojar.

Problems of inducing large coverage constraint-based dependency grammar.

*In Proceedings of the International Workshop on Constraint Solving and Language Processing, Roskilde/DK, 2004.*



Joan Bresnan and Ronald Kaplan.

Lexical-Functional Grammar: A formal system for grammatical representation.

*In Joan Bresnan, editor, The Mental Representation of Grammatical Relations, pages 173–281. The MIT Press, Cambridge/US, 1982.*

# References



Michael Collins.

*Head-Driven Statistical Models for Natural Language Parsing.*  
PhD thesis, University of Pennsylvania, 1999.



Ralph Debusmann.

*Extensible Dependency Grammar: A Modular Grammar  
Formalism Based On Multigraph Description.*  
PhD thesis, Universität des Saarlandes, 4 2006.

# References



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

-  Ralph Debusmann, Denys Duchier, and Andreas Rossberg. Modular Grammar Design with Typed Parametric Principles. In *Proceedings of FG-MOL 2005, Edinburgh/UK, 2005*.
-  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/MX, 2005*. Springer.

# References



Raph Debusmann.

A declarative grammar formalism for dependency grammar.

Diploma thesis, Saarland University, 2001.

<http://www.ps.uni-sb.de/Papers/abstracts/da.html>.



Denys Duchier and Ralph Debusmann.

Topological dependency trees: A constraint-based account of linear precedence.

In *Proceedings of ACL 2001*, Toulouse/FR, 2001.



# 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.*



Martin Forst, Nuria Bertomeu, Berthold Crysmann, Frederik Fouvry, Silvia Hansen-Schirra, and Valia Kordoni.

Towards a dependency-based gold standard for German parsers—the TiGer dependency bank.

*In Proceedings of the 5th Int. Workshop on Linguistically Interpreted Corpora, Geneva/CH, 2004.*

# References



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.



Paul Kingsbury and Martha Palmer.

From Treebank to PropBank.

In *Proceedings of LREC-2002*, Las Palmas/ES, 2002.

# References



I. Dan Melamed, Giorgio Satta, and Benjamin Wellington.  
Generalized Multitext Grammars.  
*In Proceedings of ACL 2004, Barcelona/ES, 2004.*



Mathias Möhl.  
Modellierung natürlicher Sprache mit Hilfe von Topologischer  
Dependenzgrammatik, 2004.  
Fortgeschrittenenpraktikum, Saarland University,  
<http://www.ps.uni-sb.de/rade/papers/related/Moehl04.pdf>.

# References



## Mozart Consortium.

The Mozart-Oz website, 2006.

<http://www.mozart-oz.org/>.



## Renjini Narendranath.

Evaluation of the stochastic extension of a constraint-based dependency parser, 2004.

[Bachelorarbeit, Saarland University.](#)

# References



Marwan Odeh.

Topologische Dependenzgrammatik fürs Arabische, 2004.  
Forschungspraktikum, Saarland University.



Carl Pollard and Ivan A. Sag.

*Head-Driven Phrase Structure Grammar.*  
University of Chicago Press, Chicago/US, 1994.

# References



Christian Schulte and Peter J. Stuckey.

Speeding up constraint propagation.

In *Tenth International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 619–633, Toronto/CA, 2004.

Springer-Verlag.



Christian Schulte and Guido Tack.

Views and iterators for generic constraint implementations.

In Christian Schulte, Fernando Silva, and Ricardo Rocha, editors, *Proceedings of the Fifth International Colloquium on Implementation of Constraint and Logic Programming Systems*, pages 37–48, Sitges/ES, 2005.

# References



Mark Steedman.

*The Syntactic Process.*

MIT Press, Cambridge/US, 2000.



Bonnie Webber, Aravind Joshi, Eleni Miltsakaki, Rashmi Prasad, Nikhil Dinesh, Alan Lee, and Katherine Forbes.

A short introduction to the Penn Discourse TreeBank.

Technical report, University of Pennsylvania, 2005.

# Notational Conveniences

- strict dominance:

$$v \rightarrow_d^+ v' \stackrel{\text{def}}{=} v \rightarrow_d v' \vee (\exists v'' : v \rightarrow_d v'' \wedge v \rightarrow_d^+ v'')$$



The principles of the grammar are defined as follows. First, we state that the root of the analysis must have truth value 1 (point). Second, we state that the truth value of an implication is the implication of the truth value of the antecedent (arg1) and the consequent (arg2). As implications never have bars below them, their bars value is 1. Third, we state that zeros must always have truth value zero, and that their bars value is also 1.

# Principles: Roots, Implications and Zeros

- roots:

$$\begin{aligned}
 plRoots &= \forall v : \\
 &\neg \exists v' : v' \rightarrow_{PL} v \Rightarrow (PL\ v).truth \doteq 1
 \end{aligned}$$

- implications:

$$\begin{aligned}
 plImpls &= \forall v, v', v'' : \\
 &(v \xrightarrow{arg1}_{PL} v' \wedge v \xrightarrow{arg2}_{PL} v'' \Rightarrow \\
 &(PL\ v).truth \doteq ((PL\ v').truth \Rightarrow (PL\ v'').truth)) \wedge \\
 &(PL\ v).bars \doteq 1
 \end{aligned}$$

- zeros:

$$\begin{aligned}
 plZeros &= \forall v : \\
 &(W\ v) \doteq 0 \Rightarrow \\
 &(PL\ v).truth \doteq 0 \wedge \\
 &(PL\ v).bars \doteq 1
 \end{aligned}$$

For nodes corresponding to variables, we state that their bars value must equal the bars value of their bar daughter. For nodes corresponding to bars, which have no truth value, we set the truth attribute to an arbitrary value (zero). Now if there are no more bars below, the bars value of the node is 1. If there is a bar daughter, then bars value of the mother must be one greater than the bar value of the daughter.

# Principles: Variables and Bars

- variables:

$$\begin{aligned}
 plVars &= \forall v, v' : \\
 (W v) &\doteq var \Rightarrow \\
 v \xrightarrow{\text{bar}}_{PL} v' &\Rightarrow (PL v).bars \doteq (PL v').bars
 \end{aligned}$$

- bars:

$$\begin{aligned}
 plBars &= \forall v : \\
 (W v) &\doteq I \Rightarrow \\
 (PL v).truth &\doteq 0 \wedge \\
 \neg \exists v' : v \rightarrow_{PL} v' &\Rightarrow (PL v).bars \doteq 1 \wedge \\
 (\forall v' : v \xrightarrow{\text{bar}}_{PL} v' &\Rightarrow incr v' v)
 \end{aligned}$$

This is how we can establish coreferences between the variable occurrences: we stipulate that for each pair of nodes corresponding to variable occurrences, if they have the same bars values, then their truth values must be the same.

# Principles: Coreference

- coreference:

$$\begin{aligned} plCoref &= \forall v, v' : \\ &(W\ v) \doteq \mathit{var} \wedge (W\ v') \doteq \mathit{var} \Rightarrow \\ &(PL\ v).bars \doteq (PL\ v').bars \Rightarrow (PL\ v).truth \doteq (PL\ v').truth \end{aligned}$$