Scrambling as the Combination of Relaxed Context-Free Grammars in a Model-Theoretic Grammar Formalism

Ralph Debusmann

Programming Systems Lab, Saarbrücken, Germany

MTS@10, ESSLLI 07, Trinity College, Dublin, August 15, 2007
Revised Version

Overview

Overview

MTS and the Shadow of GES

- 1996: first ESSLLI workshop on MTS
- (Pullum and Scholz 2001): (work on MTS so far) "has been done in the shadow of GES. It has largely focused on comparing MTS and GES."
- (Rogers 2004) steps out of the shadow: uses MTS to explore extensions of a GES framework (TAG)
- (Debusmann 2007 MTS): uses MTS to explore extensions of CFG, based on Extensible Dependency Grammar (XDG)

Extensible Dependency Grammar (XDG)

- model-theoretic meta grammar formalism (Debusmann 2006)
- multi-dimensional: models tuples of dependency graphs
- "meta":
  1. axiomatize your own dependency-based grammatical theory
  2. extend it
  3. prototype and verify it using the XDG Development Kit (XDK) (Debusmann, Duchier and Niehren 2004)
- extensions:
  1. add/remove constraints
  2. combine grammars (XDG closed under intersection and union)

Extending CFG

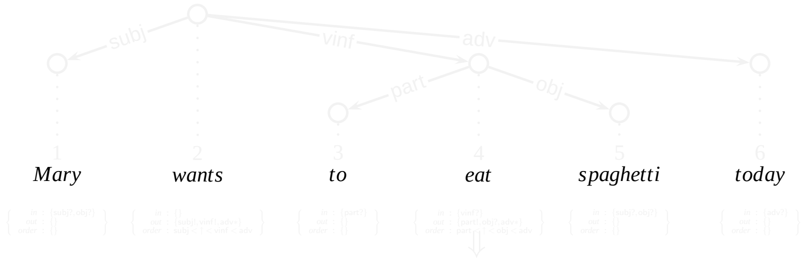
- this paper: apply some of these extensions to CFG
- starting point: modular model of lexicalized context-free grammar (LCFG) in XDG (Debusmann 2006)
- new handle on CFG:
  1. relax CFG constraints, e.g. allow discontinuous constituents
  2. combine CFGs and relaxed CFGs (e.g. intersect them)
- with this degree of extensibility: how far can we take CFG?

Overview

Dependency Graph

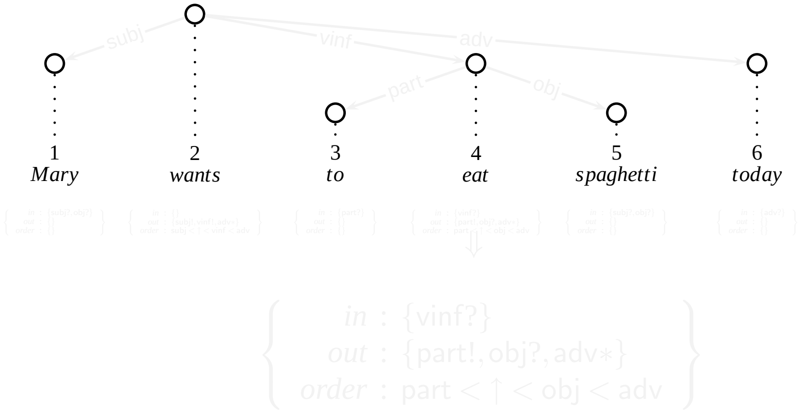
- XDG analyses: tuples of dependency graphs
- countless definitions for “dependency graph” in the literature
- how do we define it?

Dependency Graph
Words

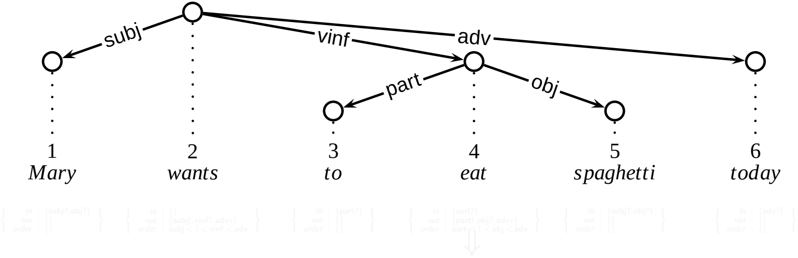


$$\left\{ \begin{array}{rcl} \textit{in} & : & \{\text{vinf?}\} \\ \textit{out} & : & \{\text{part!}, \text{obj?}, \text{adv}*\} \\ \textit{order} & : & \text{part} < \uparrow < \text{obj} < \text{adv} \end{array} \right\}$$

Dependency Graph
Nodes



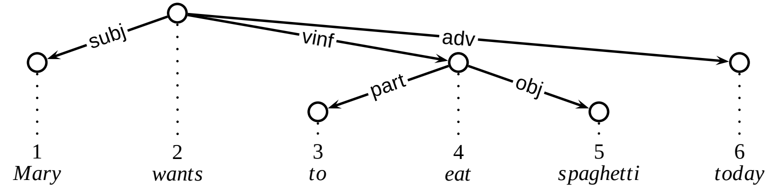
Dependency Graph
Labeled Edges



$$\left\{ \begin{array}{rcl} in & : & \{\mathsf{vinf?}\} \\ out & : & \{\mathsf{part!}, \mathsf{obj?}, \mathsf{adv*}\} \\ order & : & \mathsf{part} < \uparrow < \mathsf{obj} < \mathsf{adv} \end{array} \right\}$$

Dependency Graph

Node Attributes

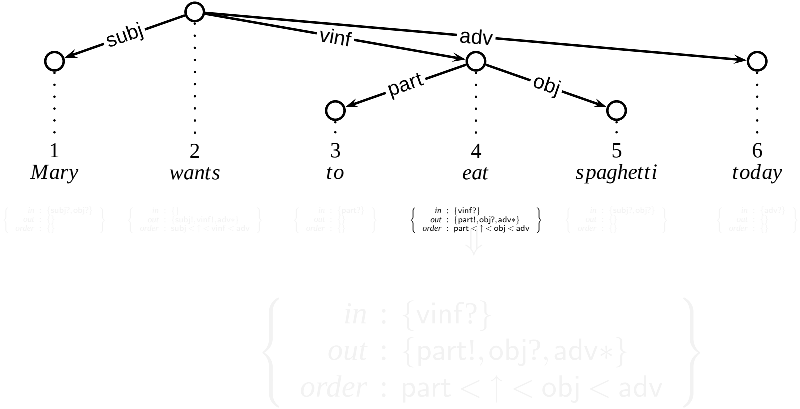


	in	*out*	*order*
Mary	{subj?, obj?}	{}	{}
wants	{}	{subj!, vinf!, adv∗}	subj < ↑ < vinf < adv
to	{part?}	{}	{}
eat	{vinf?}	{part!, obj?, adv∗}	part < ↑ < obj < adv
spaghetti	{subj?, obj?}	{}	{}
today	{adv?}	{}	{}

⇓

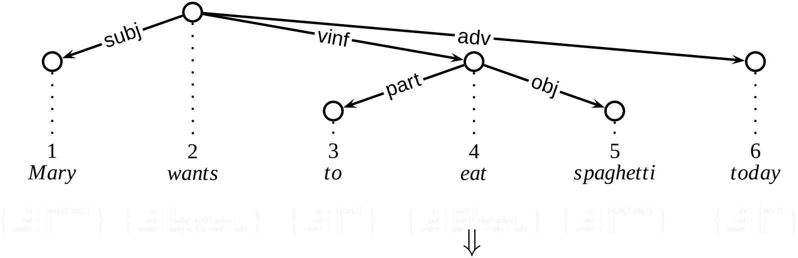
$$\left\{ \begin{array}{rcl} \textit{in} & : & \{\text{vinf?}\} \\ \textit{out} & : & \{\text{part!}, \text{obj?}, \text{adv}*\} \\ \textit{order} & : & \text{part} < \uparrow < \text{obj} < \text{adv} \end{array} \right\}$$

Dependency Graph
Node Attributes



Dependency Graph

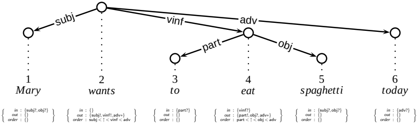
Node Attributes



$$\left\{ \begin{array}{rcl} \mathit{in} & : & \{\mathsf{vinf?}\} \\ \mathit{out} & : & \{\mathsf{part!}, \mathsf{obj?}, \mathsf{adv}*\} \\ \mathit{order} & : & \mathsf{part} < \uparrow < \mathsf{obj} < \mathsf{adv} \end{array} \right\}$$

Dependency Graph

Formal Definition

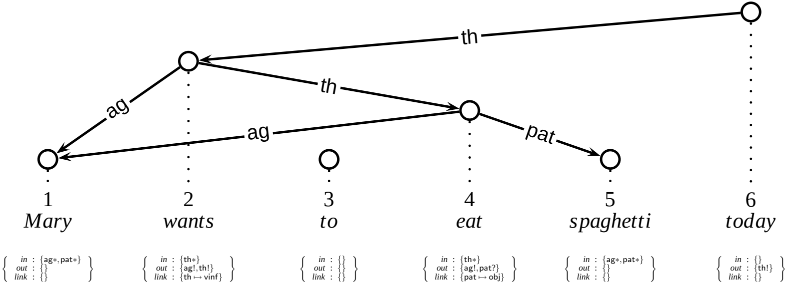


Definition

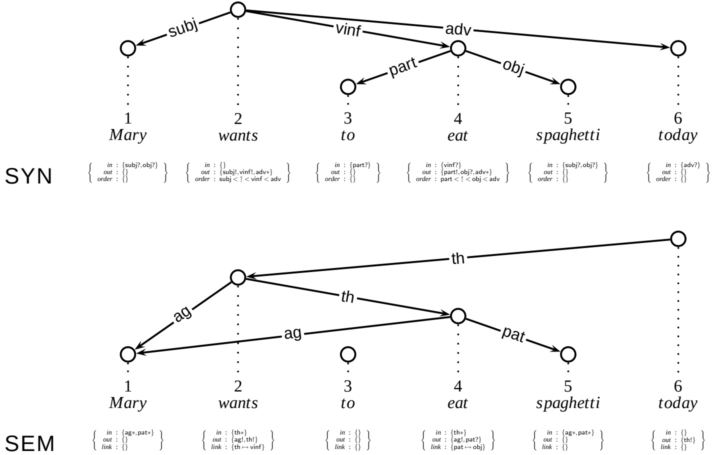
Given finite sets of edge labels L, words W, attributes A and values U, a dependency graph is a quintuple $(V, E, <, nw, na)$, where:

1. $V = \{1, \ldots, n\}$
2. $E \subseteq V \times V \times L$
3. $< \subseteq V \times V$
4. $nw \in V \to W$
5. $na \in V \to A \to U$

Semantic Dependency Graph



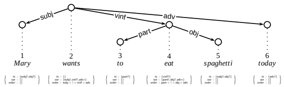
Dependency Multigraph



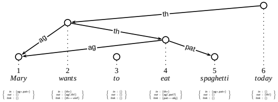
Dependency Multigraph

Formal Definition

SYN



SEM



Definition

Given L, W, A, U, and a finite set of dimensions D, a dependency multigraph is a quintuple $(V, E, <, nw, na)$, where:

1. $V = \{1, \dots, n\}$
2. $E \subseteq V \times V \times L \times D$
3. $< \subseteq V \times V$
4. $nw \in V \to W$
5. $na \in V \to D \to A \to U$

Grammar

Definition

An XDG grammar is a triple $G = (MT, lex, P)$, where:

1. MT: multigraph type (determines the dimensions, words, labels, attributes and values)
2. lex: lexicon
3. P: principles

Principles
Definition

Definition

XDG principles $\phi \in P$ are defined in a FOL:

$$t \quad ::= \quad c \mid x$$

$$\begin{array}{rcl} \phi & ::= & \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x : \phi \mid t = t' \\ & \mid & v \xrightarrow{l}_d v' \\ & \mid & v < v' \\ & \mid & w(v) = w \\ & \mid & (t_1 \ldots t_n) \in a_d(v) \end{array}$$

Principles

Transitive Closure

- FOL cannot express the transitive closure of the edge relation
- choices:
  1. go for a more expressive logic (e.g. MSO)
  2. encode it in the model, idea from XPath research e.g. (Filiot et al. 2007)
- XDG in practice: no other need to go $> FOL$, so 2.
- dependency multigraph defined over the labeled dominance relation: $(V, E^+, <, nw, na)$

Definition

$v \xrightarrow{l}_d \rightarrow^*_d v' \in E^+$ iff on d, there is an edge from v to another node v'' labeled l, and a path of $n \geq 0$ edges from v'' to v'.

Principles

Labeled Dominance Relation and Other Relations

Dominance

$$v \rightarrow_d^+ v' \stackrel{\text{def}}{=} \exists l : v \xrightarrow{l}_d \rightarrow_d^* v'$$

Labeled Edge

$$v \xrightarrow{l}_d v' \stackrel{\text{def}}{=} v \xrightarrow{l}_d \rightarrow_d^* v' \wedge \neg\exists v'' : v \rightarrow_d^+ v'' \wedge v'' \rightarrow_d^+ v'$$

Edge

$$v \rightarrow_d v' \stackrel{\text{def}}{=} \exists l : v \xrightarrow{l}_d v'$$

Principles

Definition (revised)

Definition

XDG principles $\phi \in P$ are defined in a FOL:

$$t \quad ::= \quad c \mid x$$

$$\begin{array}{rcl} \phi & ::= & \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x : \phi \mid t = t' \\ & \mid & v \xrightarrow{l}_d \rightarrow^*_d v' \\ & \mid & v < v' \\ & \mid & w(v) = w \\ & \mid & (t_1 \ldots t_n) \in a_d(v) \end{array}$$

Principles

Examples

- predefined e.g.:
  - tree
  - DAG (directed acyclic graph)
  - projectivity
  - valency
  - order
  - linking
- easy to define new principles:
  1. only knowledge of FOL required
  2. can immediately be prototyped and verified in the XDG Development Kit

Models

Definition

The set of models $m\ G$ of a grammar $G = (MT, lex, P)$ contains all multigraphs M which:

1. have multigraph type MT
2. satisfy the lexicon lex
3. satisfy the conjunction of the principles in P

String Language

Definition

The string language $L\ G$ of an XDG grammar G is the set of strings of its models:

$$L\ G \quad = \quad \{nw\ 1 \ldots nw\ |V| \mid (V, E^+, <, nw, na) \in m\ G\}$$

Closure Properties

- proven in (Debusmann 2007 MTS): string languages licensed by XDG grammars closed under:
  - intersection
  - union
- proof idea: given two grammars G_1 and G_2 with disjoint dimensions and defined over same set of words:
  1. union their dimensions, labels, attributes and values
  2. multiply out their lexicons
  3. combine the conjunction of their principles with $\wedge$ (intersection), $\vee$ (union)

Recognition Problems

- given a grammar G and a string s, is s in $L\ G$?
- complexity (Debusmann 2007 FO):
  - universal recognition problem: both G and s are variable: PSPACE-complete
  - fixed recognition problem: G is fixed and s is variable: NP-complete
  - instance recognition problem: the principles are fixed, and the lexicon and s are variable: NP-complete
- specific instances of XDG (e.g. LCFG) can be less complex

Parsing Problem

- given a grammar G and an input string $s = a_1 \ldots a_n$, find all $M = (V, E^+, <, nw, na) \in m\ G$ such that:
  1. $V = \{1, \ldots, n\}$
  2. $nw = \{i \mapsto a_i \mid 1 \leq i \leq n\}$
  3. $< = \{(v, v') \mid v < v'\}$
- input string completely determines the set of nodes, only finite number of edges between nodes added, but no nodes!
- “fixed size property”: efficient parsing of XDG grammars using constraint programming (Schulte 2002)

Overview

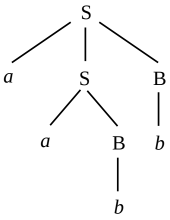
LCFG in XDG

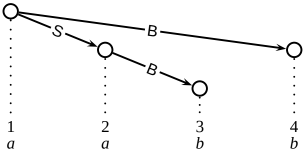
- LCFG recap:
  - an LCFG is a CFG where each rule has precisely one terminal symbol on its right hand side
  - LCFG corresponds directly to projective dependency grammar (Gaifman 1965), (Kuhlmann 2007)
- (Debusmann 2006): model-theoretic axiomatization of LCFG in XDG based on (McCawley 1968)

Axiomatization

Idea

- derivation trees of LCFG correspond directly to projective dependency trees in XDG
- example:



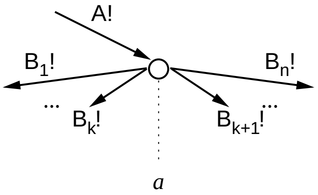


Axiomatization
Principles

- XDG model of LCFG uses four principles:
  1. tree
  2. projectivity
  3. valency
  4. order
- lexical entries for the valency and order principles model the production rules of the LCFG

Axiomatization
Production Rules

- each LCFG production rule corresponds to a lexical entry in XDG
- lexical entry constrains:
  - incoming/outgoing edges
  - order of the outgoing edges



$$A \longrightarrow B_1 \ldots B_k a B_{k+1} \ldots B_n$$

Overview

Scrambling

- theory of topological fields to describe German word order (Herling 1821), (Erdmann 1886):
  1. verbs positioned in the "verb-cluster" at the right end
  2. verbs preceded by the non-verbal dependents in the "Mittelfeld"
  3. scrambling: elements of the Mittelfeld can be freely permuted
- example:

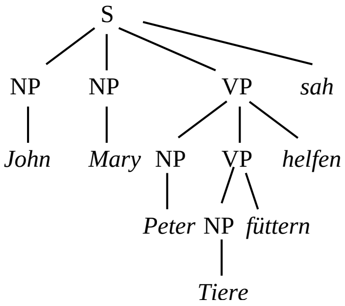
Mittelfeld					verb cluster		
(dass)	*John*$_1$	*Mary*$_1$	*Peter*$_2$	*Tiere*$_3$	*füttern*$_3$	*helfen*$_2$	*sah*$_1$
(that)	*John*$_1$	*Mary*$_1$	*Peter*$_2$	*animals*$_3$	*feed*$_3$	*help*$_2$	*saw*$_1$

LCFG

- LCFG G_{ID} modeling the example:

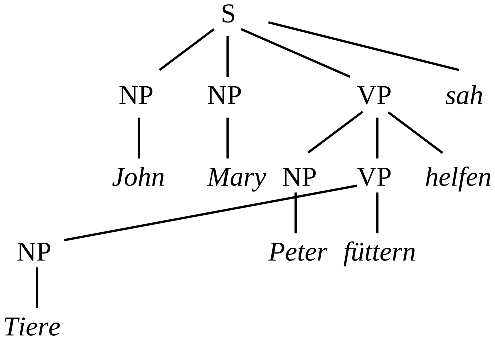
S	→	NP NP VP *sah*	VP	→	NP VP *helfen*
VP	→	NP *füttern*	NP	→	*John*
NP	→	*Mary*	NP	→	*Peter*
NP	→	*Tiere*			

- example analysis:



Discontinous Analyses

- G_{ID} undergenerates: does not allow NPs in the Mittelfeld to occur in more than one permutation
- does not license discontinuous analyses such as:



- what can we do now? CFGs cannot model discontinuous analyses...

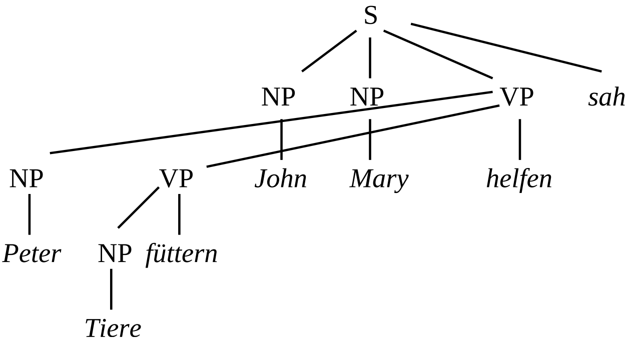
First Idea

Relax the LCFG

- first idea:
  1. axiomatize the LCFG G_{ID} in XDG
  2. use the additional expressive power in XDG to allow discontinuous constituents, by dropping the projectivity principle

Relaxed LCFG

- problem: overgeneration, e.g. also licenses:



Second Idea

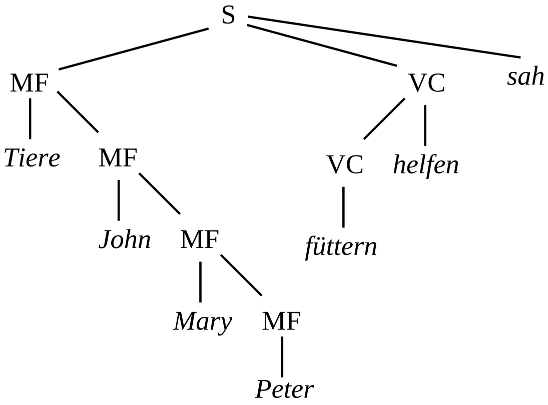
Topological LCFG

- second idea: create a new, topological LCFG called G_{LP} in the spirit of topological fields theory (Kathol 1995), (Gerdes and Kahane 2001), (Duchier and Debusmann 2001)
- G_{LP} orders all NPs to the left of the verbs:

$$
\begin{array}{rcl}
\text{S} & \rightarrow & \text{MF VC } \textit{sah} \\
\text{VC} & \rightarrow & \textit{füttern} \\
\text{MF} & \rightarrow & \textit{John} \text{ MF} \\
\text{MF} & \rightarrow & \textit{Mary} \text{ MF} \\
\text{MF} & \rightarrow & \textit{Peter} \text{ MF} \\
\text{MF} & \rightarrow & \textit{Tiere} \text{ MF}
\end{array}
\qquad
\begin{array}{rcl}
\text{VC} & \rightarrow & \text{VC } \textit{helfen} \\
\text{MF} & \rightarrow & \textit{John} \\
\text{MF} & \rightarrow & \textit{Mary} \\
\text{MF} & \rightarrow & \textit{Peter} \\
\text{MF} & \rightarrow & \textit{Tiere}
\end{array}
$$

Topological LCFG Analysis

- example analysis:



Topological LCFG Review

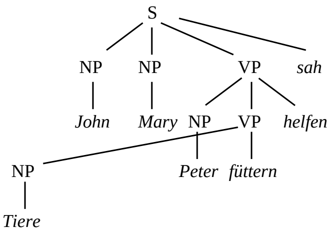
- G_{LP} does license the correct string language
- problem: G_{LP} loses the syntactic dependencies between the verbs and their non-verbal dependents
- renders grammar practically useless: impossible to get from a G_{LP} analysis to the semantics of a sentence

Third Idea
Intersection

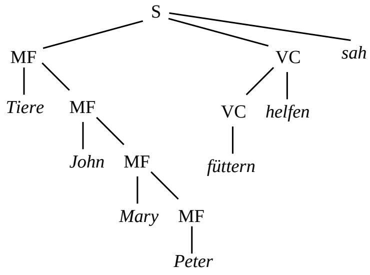
- original LCFG: undergenerated
- ideas for remedying:
  1. axiomatize G_{ID} in XDG and relax it: overgeneration
  2. topological LCFG G_{LP}: essential syntactic dependencies lost
- third idea: axiomatize both G_{ID} and G_{LP} in XDG, and use the additional expressive power to intersect them!
- two grammars "help out" each other:
  1. G_{LP}: avoids overgeneration
  2. G_{ID}: still represents the essential syntactic dependencies

Example ID/LP Analysis

- example analysis:



ID



LP

Overview

Summary

- introduced model-theoretic meta grammar formalism of Extensible Dependency Grammar (XDG)
- in XDG, any dependency-based grammar formalism can be axiomatized model-theoretically
- once axiomatized, it can easily be extended
- using an axiomatization of CFG, we have explored:
  1. the relaxation of the CFG contiguity criterion
  2. the intersection of CFGs and relaxed CFGs
- lead us to a model of scrambling, one of the most complicated phenomena in syntax, as the combination of two grammars formulated in one of the simplest of all grammar formalisms

Beyond CFG

- also axiomatized in XDG:
  - TAG (Joshi 1987), axiomatization: (Debusmann 2007 (unpublished))
  - Dominance Constraints (Egg et al. 2001), axiomatization: (Debusmann 2006)
  - Polarized Unification Grammars (PUG) (Kahane 2006), axiomatization: (Lison 2006)
- once axiomatized: can freely combine them!
- combine TAG (for syntax) and Dominance Constraints (for semantics) etc.

Blatant Advertisement

- interested? why not pick your own favorite grammar formalism, and:
  1. axiomatize it
  2. extend it
  3. combine it with other formalisms
- XDG homepage: just look for “xdg” with Google
  - papers
  - talks
  - ESSLLI 2004 course
  - mailing list
  - development kit

Thanks for your attention!

References I

Pierre Boullier.
Range Concatenation Grammars.
In *Proceedings of IWPT 2000*, Trento/IT, 2000.

David Chiang.
Uses and abuses of intersected languages.
In *Proceedings of TAG+7*, pages 9–15, Vancouver/CA, 2004.

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

Ralph Debusmann.
The complexity of First-Order Extensible Dependency Grammar.
Technical report, Saarland University, 2007.

References II

Ralph Debusmann, Denys Duchier, and Joachim Niehren.
The XDG grammar development kit.
In *Proceedings of the MOZ04 Conference*, Charleroi/BE, 2004.

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

Markus Egg, Alexander Koller, and Joachim Niehren.
The Constraint Language for Lambda Structures.
Journal of Logic, Language, and Information, 2001.

O. Erdmann.
Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung dargestellt.
Erste Abteilung, Stuttgart/DE, 1886.

References III

Emmanuel Filiot, Joachim Niehren, Jean-Marc Talbot, and Sophie Tison.
Polynomial time fragments of xpath with variables.
In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Beijing/CN, 2007.

Haim Gaifman.
Dependency systems and phrase-structure systems.
Information and Control, 8(3):304–337, 1965.

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

S.H.A. Herling.
Über die Topik der deutschen Sprache, 1821.

References IV

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.

Sylvain Kahane.
Polarized unification grammars.
In *Proceedings of ACL 2006*, pages 137–144, Sydney/AU, 2006.

Andreas Kathol.
Linearization-Based German Syntax.
PhD thesis, Ohio State University, Ohio/US, 1995.

Marco Kuhlmann.
Drawings as Models of Syntactic Structure.
PhD thesis, Universität des Saarlandes, 8 2007.

References V

Pierre Lison.
Implémentation d'une interface sémantique-syntaxe basée sur des grammaires d'unification polarisées.
Master's thesis, Univesité Catholique de Louvain, 2006.

J. D. McCawley.
Concerning the base component of a Transformational Grammar.
Foundations of Language, 4:243–269, 1968.

I. Dan Melamed.
Multitext grammars and synchronous parsers.
In *Proceedings of HLT-NAACL 2003* Edmonton/CA, 2003.

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

References VI

Geoffrey K. Pullum and Barbara C. Scholz.
On the distinction between model-theoretic and generative-enumerative syntactic frameworks.
Logical Aspect of Computational Linguistics: 4th International Conference, Berlin/DE, 2001.

James Rogers.
On scrambling, another perspective.
In *Proceedings of TAG+7*, Vancouver/CA, 2004.

Christian Schulte.
Programming Constraint Services, volume 2302 of *Lecture Notes in Artificial Intelligence*.
Springer-Verlag, 2002.

Tree Principle

- four conditions:
  1. there must be no cycles
  2. there is precisely one node without a mother (the root)
  3. all nodes have zero or one mothers
  4. all differently labeled subtrees must be disjoint

Definition

$$\begin{array}{l} tree_d = \\ \forall v : \neg(v \rightarrow_d^+ v) \wedge \\ \exists! v : \neg \exists v' : v' \rightarrow_d v \wedge \\ \forall v : ((\neg \exists v' : v' \rightarrow_d v) \vee (\exists! v' : v' \rightarrow_d v)) \wedge \\ \forall v : \forall v' : \forall l : \forall l' : v \xrightarrow{l}_d \rightarrow_d^* v' \ \wedge \ v \xrightarrow{l'}_d \rightarrow_d^* v' \ \Rightarrow \ l = l' \end{array}$$

Projectivity Principle

- forbids crossing edges by stipulating that all nodes positioned between a head and a dependent must be below the head

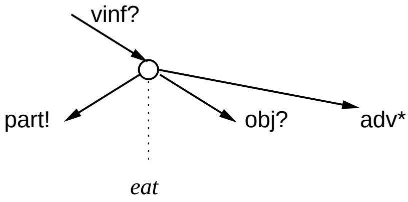
Definition

$$
\begin{aligned}
& projectivity_d = \\
& \forall v, v' : \\
& (v \rightarrow_d v' \;\wedge\; v < v' \Rightarrow \forall v'' : v < v'' \wedge v'' < v' \Rightarrow v \rightarrow_d^+ v'') \wedge \\
& (v \rightarrow_d v' \;\wedge\; v' < v \Rightarrow \forall v'' : v' < v'' \wedge v'' < v \Rightarrow v \rightarrow_d^+ v'')
\end{aligned}
$$

Valency Principle

Intuition

- lexically constrains the incoming and outgoing edges of each node on a dimension d
- graphical lexical entry:



Valency Principle

Lexical Attributes

- attributes and types, given set of labels $L = dl\ d$:

$$\left\{ \begin{array}{rcl} in & : & 2^{L \times \{!,+,?,*\}} \\ out & : & 2^{L \times \{!,+,?,*\}} \end{array} \right\}$$

- example:

$$\left\{ \begin{array}{rcl} in & : & \{(\mathsf{vinf}, ?)\} \\ out & : & \{(\mathsf{part}, !), (\mathsf{obj}, ?), (\mathsf{adv}, *)\} \end{array} \right\}$$

- syntactic sugar:

$$\left\{ \begin{array}{rcl} in & : & \{\mathsf{vinf}?\} \\ out & : & \{\mathsf{part}!, \mathsf{obj}?, \mathsf{adv}*\} \end{array} \right\}$$

Valency Principle

Definition

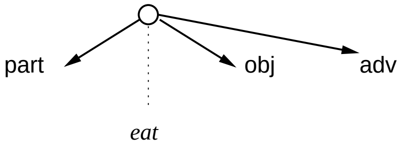
Definition

$$
\begin{array}{l}
valency_d = \\
\forall v : \forall l : \\
((l,!) \in in_d(v) \;\Rightarrow\; \exists! v' : v' \xrightarrow{l}_d v) \;\wedge \\
((l,+) \in in_d(v) \;\Rightarrow\; \exists v' : v' \xrightarrow{l}_d v) \;\wedge \\
((l,?) \in in_d(v) \;\Rightarrow\; \neg\exists v' : v' \xrightarrow{l}_d v \;\vee\; \exists! v' : v' \xrightarrow{l}_d v) \;\wedge \\
(\neg(l,!) \in in_d(v) \;\wedge\; \neg(l,+) \in in_d(v) \;\wedge\; \neg(l,?) \in in_d(v) \;\wedge \\
\;\neg(l,*) \in in_d(v) \;\Rightarrow\; \neg\exists v' : v' \xrightarrow{l}_d v) \;\wedge \\
((l,!) \in out_d(v) \;\Rightarrow\; \exists! v' : v \xrightarrow{l}_d v') \;\wedge \\
\ldots
\end{array}
$$

Order Principle

Intuition

- lexically constrains the order of the outgoing edges of each node on a dimension d
- graphical lexical entry:



Order Principle

Lexical Attributes

- attribute and type, given set of labels $L = dl\ d$

$$\left\{ \begin{array}{lcl} order & : & 2^{L \times L} \end{array} \right\}$$

- example:

$$\left\{ \begin{array}{lcl} order & : & \{(\mathsf{part}, \uparrow), (\mathsf{part}, \mathsf{obj}), \\ & & (\mathsf{part}, \mathsf{adv}), (\uparrow, \mathsf{obj}), \\ & & (\uparrow, \mathsf{adv}), (\mathsf{obj}, \mathsf{adv})\} \end{array} \right\}$$

- syntactic sugar:

$$\left\{ \begin{array}{lcl} order & : & \mathsf{part} < \uparrow < \mathsf{obj} < \mathsf{adv} \end{array} \right\}$$

Order Principle
Definition

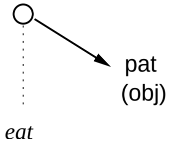
Definition

$$
\begin{array}{l}
order_d = \\
\forall v : \forall v' : \neg v \xrightarrow{\uparrow}_d v' \wedge \\
\forall v : \forall l : \forall l' : (l, l') \in order_d(v) \Rightarrow \\
(l = \uparrow \Rightarrow \forall v' : v \xrightarrow{l'}_d v' \Rightarrow v < v') \wedge \\
(l' = \uparrow \Rightarrow \forall v' : v \xrightarrow{l}_d v' \Rightarrow v' < v) \wedge \\
(\forall v' : \forall v'' : v \xrightarrow{l}_d v' \wedge v \xrightarrow{l'}_d v'' \Rightarrow v' < v'')
\end{array}
$$

Linking Principle

Intuition

- lexically constrains the realization of dependents on a dimension d_1 on another dimension d_2
- graphical lexical entry:



Linking Principle

Lexical Attributes

- attribute and type, given set of labels $L_1 = dl\ d_1$ and $L_2 = dl\ d_2$:

$$\left\{\ link\ :\ 2^{L_1 \times L_2}\ \right\}$$

- example:

$$\left\{\ link\ :\ \{(\mathsf{pat}, \mathsf{obj})\}\ \right\}$$

- syntactic sugar:

$$\left\{\ order\ :\ \{\mathsf{pat} \mapsto \mathsf{obj}\}\ \right\}$$

Linking Principle

Definition

Definition

$$linking_{d_1,d_2} =$$
$$\forall v : \forall v' : \forall l : \forall l' :$$
$$v \xrightarrow{l}_{d_1} v' \ \wedge \ (l, l') \in link_{d_1}(v) \ \Rightarrow \ v \xrightarrow{l'}_{d_2} v'$$

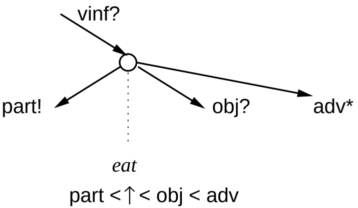
Lexical Entry

- lexical entry for "eat":

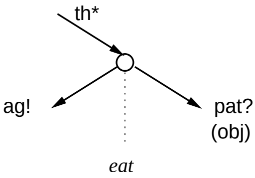
$$\textit{eat} \mapsto \left\{ \left[\begin{array}{ll} \text{SYN} : & \left[\begin{array}{rcl} \mathit{in} & : & \{\mathsf{vinf?}\} \\ \mathit{out} & : & \{\mathsf{part!}, \mathsf{obj?}, \mathsf{adv*}\} \\ \mathit{order} & : & \mathsf{part} < \uparrow < \mathsf{obj} < \mathsf{adv} \end{array} \right] \\ \text{SEM} : & \left[\begin{array}{rcl} \mathit{in} & : & \{\mathsf{th*}\} \\ \mathit{out} & : & \{\mathsf{ag!}, \mathsf{pat?}\} \\ \mathit{link} & : & \{\mathsf{pat} \mapsto \mathsf{obj}\} \end{array} \right] \end{array} \right], \ldots \right\}$$

Graphical Lexical Entry

- graphical lexical entry for “eat”:



SYN



SEM

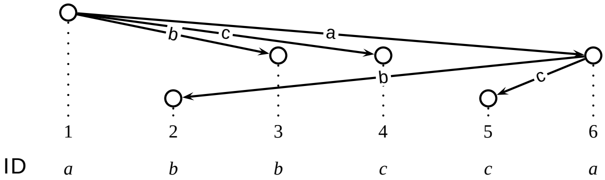
Grammar 1

Language, Example Analysis

- equally many *a*s, *b*s and *c*s in any order:

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

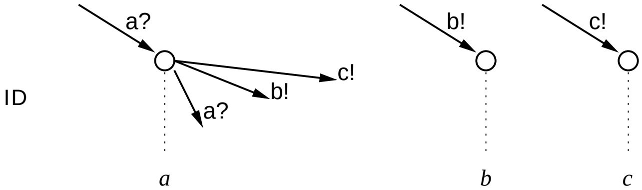
- one dimension: ID ("immediate dominance"):



Grammar 1

Principles, Lexicon

- uses tree and valency principles
- lexical entries for valency principle:



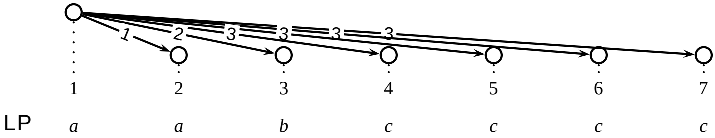
Grammar 2

Language, Example Analysis

- arbitrary many *a*s followed by arbitrary many *b*s followed by arbitrary many *c*s:

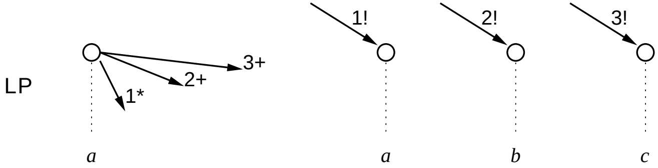
$$L_2 \quad = \quad a^+ b^+ c^+$$

- one dimension: LP (“linear precedence”):



Grammar 2
Principles, Lexicon

- uses tree, valency and order principles
- lexical entries for valency and order principles:



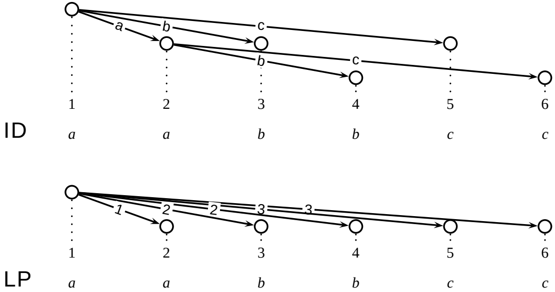
Grammar 3

Language, Example Analysis

- intersection of G_1 and G_2:

$$L_3 = L_1 \cap L_2 = \{s \in a^n b^n c^n \mid n \geq 1\}$$

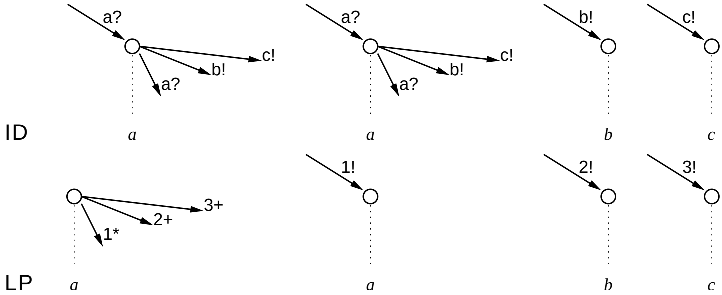
- models: multigraphs with two dimensions (ID and LP):



Grammar 3

Principles, Lexicon

- combines the principles of G_1 and G_2:
  1. ID: tree, valency
  2. LP: tree, projectivity, valency, order
- lexicon: product of the lexicons of G_1 and G_2:



Scrambling in Range Concatenation Grammars

- (Boullier 2000): structures generated by the two combined grammars are correlated only by their yields
- (Chiang 2004): only constrains the tail end of otherwise independent parallel processes ("weak parallelism")
- not enough control: treatment of scrambling in (Boullier 2000) must rely on nonexistent information in the surface string.

Extensible Dependency Grammar

- more fine-grained control:
  1. dimensions of XDG are synchronized by the input string and the corresponding nodes (shared among all dimensions)
  2. allows to stipulate any number of additional constraints to correlate the two intersected grammars
- linking constraints could be used to synchronize the rules of the two combined CFGs a la Multitext grammars (Melamed 2003), (Melamed et al. 2004)