Making linguistic dimensions autonomous: The new grammar formalism of Extensible Dependency Grammar

Ralph Debusmann

rade@ps.uni-sb.de

Programming Systems Lab

Saarland University, Saarbrücken

Overview

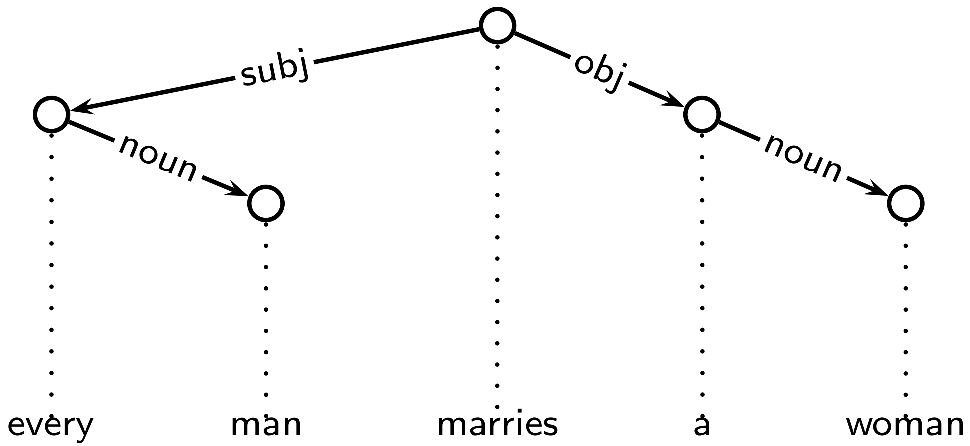
1. Motivation
2. XDG grammar formalism
3. One-dimensional principles
4. Multi-dimensional principles
5. XDG parser system
6. Conclusion

The goal

- starting point: Topological Dependency Grammar (TDG) (Duchier and Debusmann ACL 2001)
- goal: develop a *concurrent* syntax-semantics interface for TDG
- concurrency: syntax and semantics processed simultaneously
- why concurrency: allow disambiguation to happen from semantics to syntax, not only from syntax to semantics
- side-effect: dimensions of linguistic description become more autonomous

A naive syntax-semantics interface

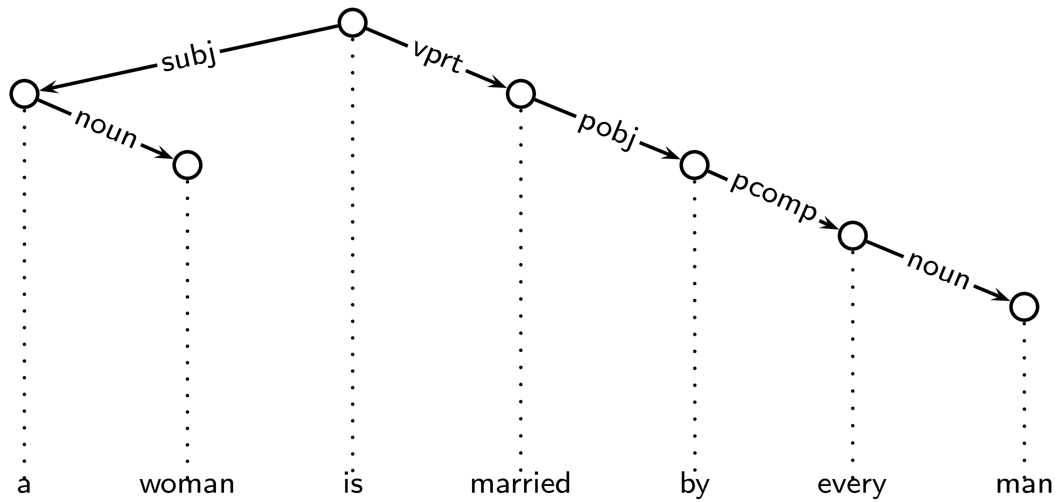
- simple dependency tree:



- directly reflects semantic predicate-argument structure: the subject *every man* is the first argument of *marries*, and the object *a woman* the second.
- function to get the semantics of the sentence is easy

Problems of a naive syntax-semantics interface

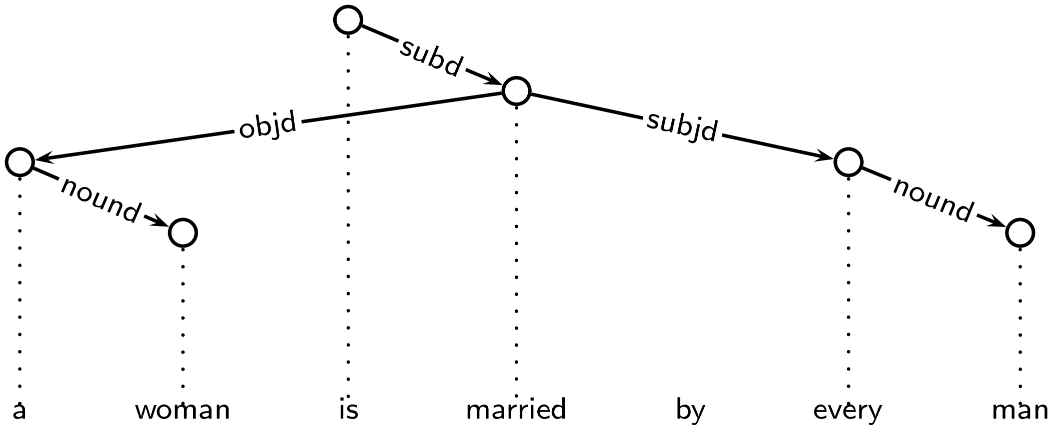
- more complicated dependency tree:



- does not directly reflect semantic predicate-argument structure
- function to get the semantics of the sentence becomes more complicated

Deep syntax

- idea from Lexical Functional Grammar (LFG) (Bresnan and Kaplan 1982)
- add a new dimension of representation: f-structure, or deep syntactic dependency graph:



- again directly reflects semantic predicate-argument structure: the deep subject *every man* is the first argument of *marries*, and the deep object *a woman* the second.
- function to get the semantics of the sentence becomes again easy

No concurrency

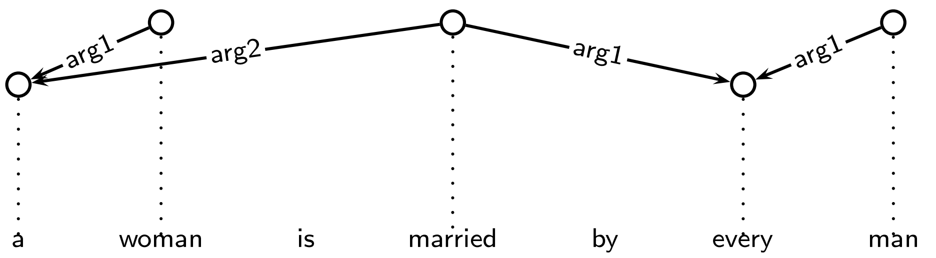
- we derive the semantics by a function from the deep syntax
- but that means that we have a *sequential* architecture: the syntax must be ready before semantics construction can begin
- what we wanted was a *concurrent* architecture

Getting concurrency

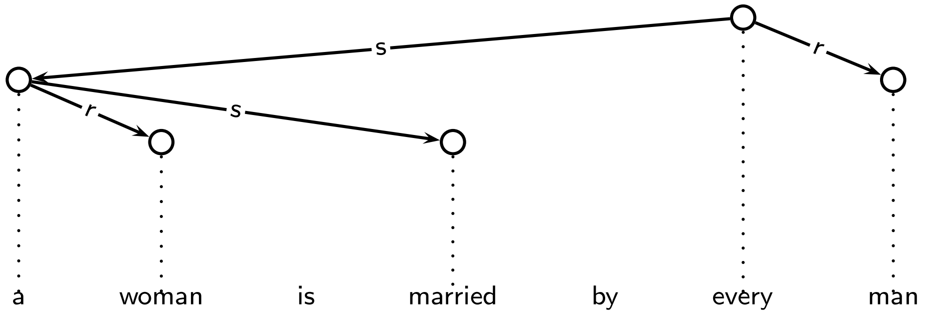
- for a concurrent architecture, the correspondence between syntax and semantics must be specified by *relations*, not by functions
- idea: introduce additional dimensions to represent semantics proper, not only syntax and deep syntax
- side-effect: syntactic and semantic dimensions become more autonomou, i.e. semantic dimensions are not just substrates of the syntactic dimensions but stands on its own

Semantic dimensions

- we introduce two semantic dimensions: predicate-argument structure (PA), and scope (SC).
- example PA dag



- example SC tree:



$$\forall x.\textit{man}(x) \Rightarrow (\exists y.\textit{woman}(y) \wedge \textit{marry}(x, y))$$

Extensible Dependency Grammar (XDG)

- XDG: new meta grammar formalism for dependency grammar (Debusmann Nancy 2003)
- generalization of Topological Dependency Grammar (TDG) (Duchier and Debusmann ACL 2001)
- arbitrary number of dimensions which correspond to graphs
- arbitrary principles on these dimensions
- XDG parser system (Debusmann Nancy 2003)

XDG instance

- XDG is actually a meta grammar formalism, i.e. it must be instantiated before use
- $Inst = (Dim, Lab, Fea, Val)$
- $Dim = \{d_1, \ldots, d_n\}$ set of *dimension identifiers*
- $Lab = L_{d_1} \cup \ldots \cup L_{d_n}$ sets of *labels* for the dimensions
- Fea set of *feature identifiers*
- Val set of *feature values*

XDG analysis

- an XDG analysis consists of a graph for each dimension
- all dimensions share the same set of nodes, but have different edges
- feature assignments to nodes parametrize the well-formedness conditions (e.g. valency)
- $A = (V, E, F)$
- V set of *nodes*
- $E \subseteq Dim \to V \times Lab \times V$ set of labeled *edges* for the dimensions
- $F \in V \to Dim \to Fea \to Val$ set of *feature assignments* to the nodes

XDG lexicon

- recap: $F \in V \to Dim \to Fea \to Val$ set of *feature assignments* to the nodes
- the set of feature assignments available to an analysis is specified by the *lexicon*
- $Lex \subseteq Dim \to Fea \to Val$

XDG constraints

- XDG well-formed conditions specified by *principles* and *input constraints*
- both define subsets of the set of all analyses
- principles: grammar-specific
- input constraints: application-specific
  - parsing: assign nodes to words and their positions in the input string
  - generation: assign nodes to semantic literals

XDG grammar

- a grammar defines an XDG instance, a set of principles, and a lexicon
- $G = (Inst, Prin, Lex)$
- given a grammar, a fixed number $m \in \mathbb{N}$, and a set of input constraints Inp, an XDG analysis $A = (V, E, F)$ is well-formed if:
  - $V = \{v_1, \ldots, v_m\}$
  - $A \in Prin \cap Inp$
  - $\forall v \in V : F(v) \in Lex$

XDG principles

- XDG principles can be *one-dimensional* or *multi-dimensional*
- one-dimensional principles: tree, directed acyclic graph, valency
- multi-dimensional principles (relational constraints between dimensions): linking, contra-dominance
- ongoing research: what precisely are possible principles?

Example XDG grammar

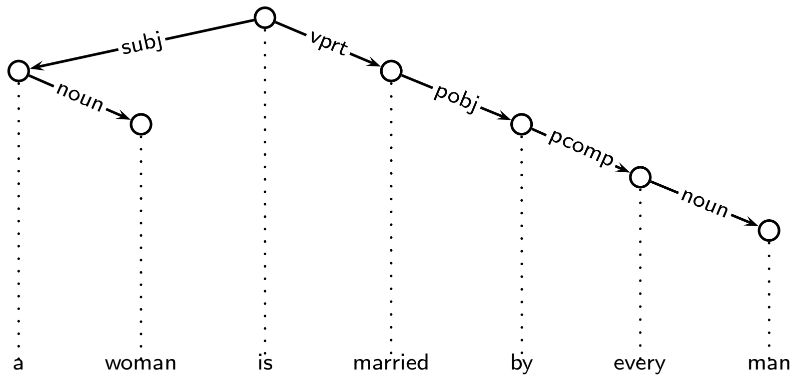
- we explain some of the most important XDG principles by an example grammar
- the example grammar is five-dimensional: Immediate Dominance (ID), Linear Precedence (LP), Deep Syntax (DS), Predicate-Argument structure (PA), and SCope structure (SC)
- ID and LP like in TDG (LP of no concern in this talk)

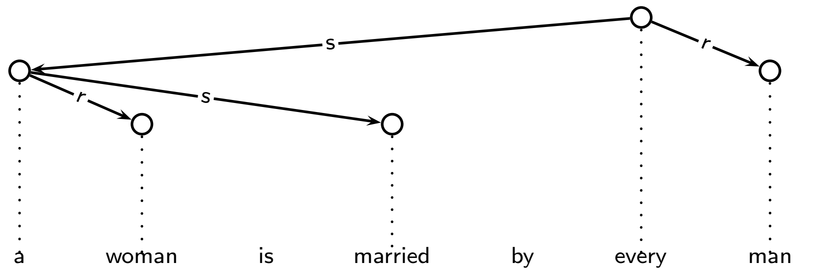
One-dimensional principles

- higher degree of modularity: restrict only one dimension at a time
- examples:
  - Tree
  - Directed Acyclic Graph
  - Valency

Tree principle

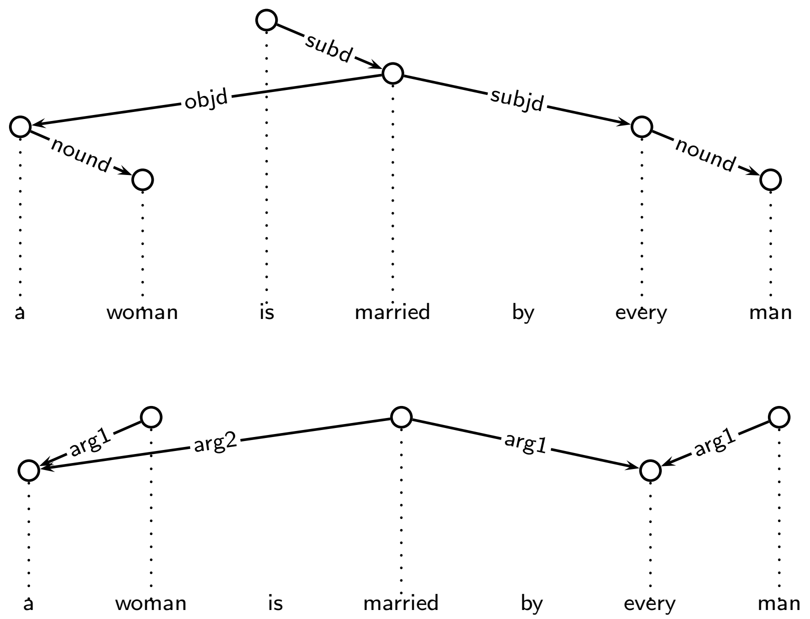
- used on the ID, (LP) and SC dimensions:





Directed acyclic graph principle

- used on the DS and PA dimensions:

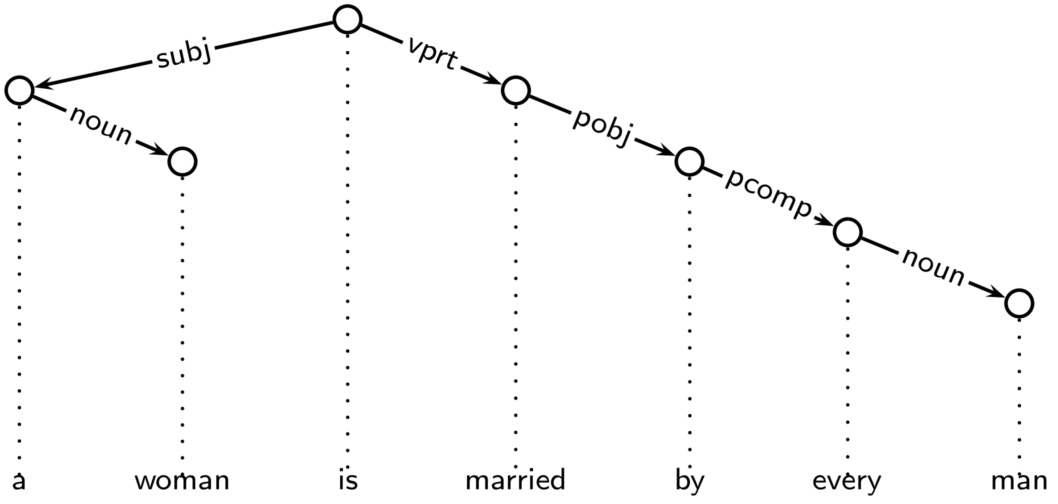


Valency

- each node has two features in and out
- in specifies the licensed incoming edges
- out specifies the licensed outgoing edges

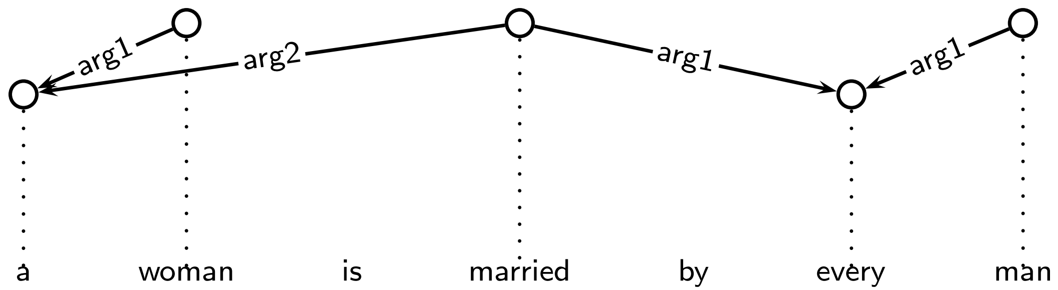
Valency example 1

- e.g. on the ID dimension, *married* is a past participle verbal complement... $in(married) = \{\text{vprt}\}$
- ... and requires a prepositional object: $out(married) = \{\text{pobj}\}$
- example ID tree:



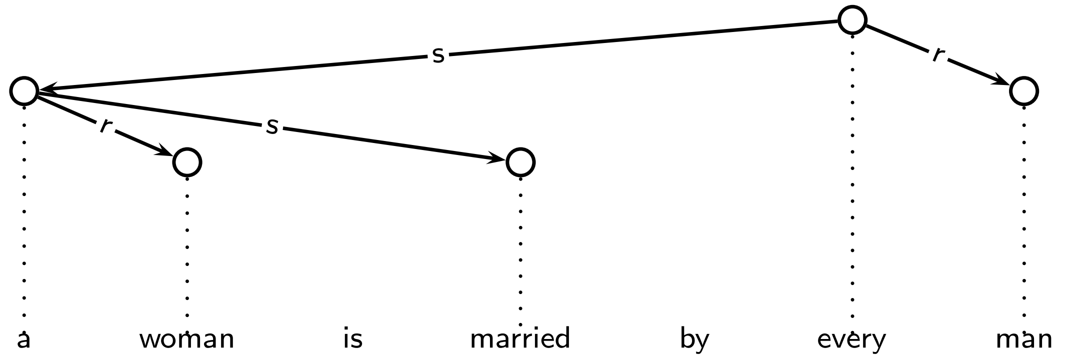
Valency example 2

- e.g. on the PA dimension, *married* can only be the root...
  $in(married) = \{\}$
- ... and requires two arguments: $out(married) = \{\text{arg1}, \text{arg2}\}$
- example PA dag:



Valency example 3

- e.g. on the SC dimension, *every* can be in the restriction or scope of another node or it can be root: $in(every) = \{\mathsf{r}?, \mathsf{s}?\}$
- ... and requires a restriction and a scope: $out(every) = \{\mathsf{r}, \mathsf{s}\}$
- example SC tree:

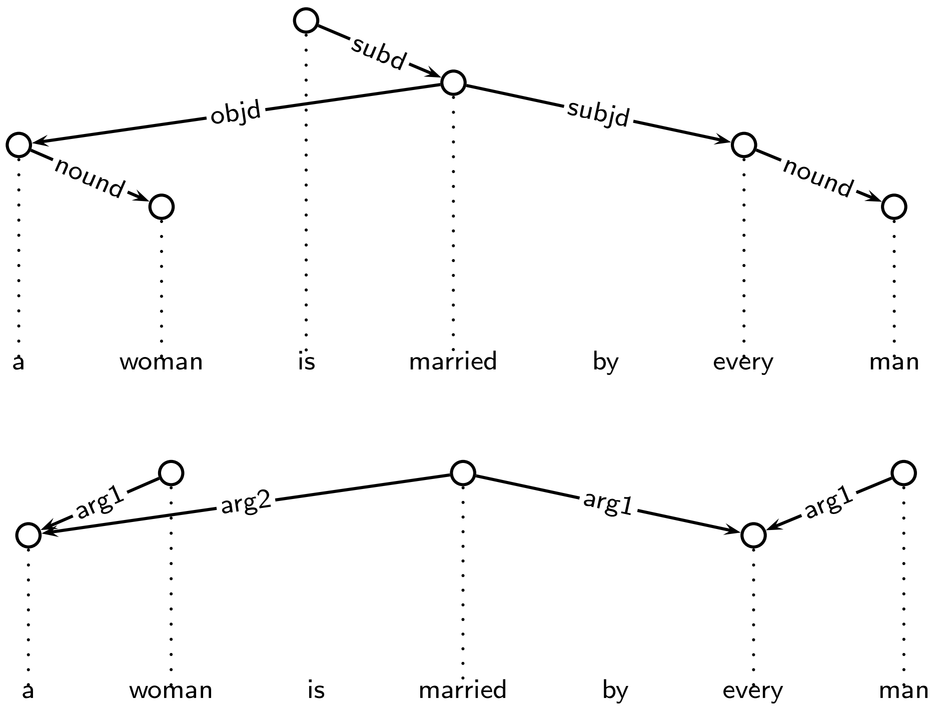


Multi-dimensional principles

- written as Horn clauses with the following predicates:
  - $v \xrightarrow{l}_{d_i} v'$ edge from v to v' labeled l on dimension d_i
  - $v \xrightarrow{l} \rightarrow^*_{d_i} v'$ edge from v labeled l, and zero or more edges to v' on dimension d_i
  - $v \xrightarrow{l} \rightarrow^* \xrightarrow{l'}_{d_i} v'$ edge from v labeled l, zero or more edges, and an edge to v' labeled l' on dimension d_i
- examples:
  - direct linking
  - indirect linking
  - contra-dominance

Direct linking principle example

- example DS dag and PA dag:



- direct linking ensures that argument 1 is realized by the deep subject: $married \xrightarrow{\text{arg1}}_{\text{PA}} every \quad \Rightarrow \quad married \xrightarrow{\text{subjd}}_{\text{DS}} every$

Direct linking principle

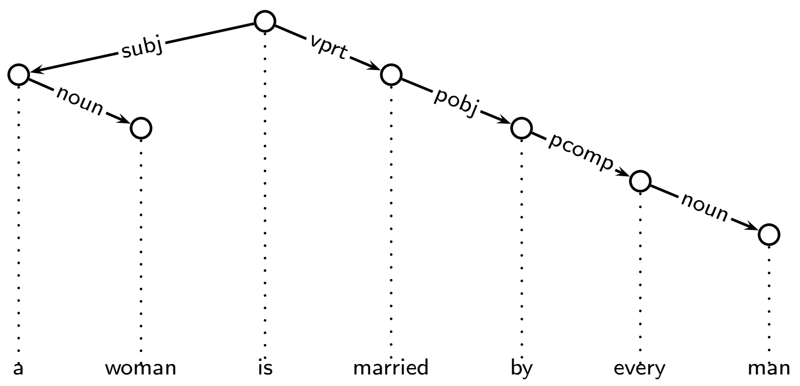
- direct linking principle in a more general form:
  $v \xrightarrow{\text{arg1}}_{\text{PA}} v' \quad \Rightarrow \quad v \xrightarrow{\text{subjd}}_{\text{DS}} v'$
- but we do not want the principle to hold for all edges
- idea: use features to restrict the direct linking principle only to a subset of the edges...
- ... and specify this information in the lexicon, e.g.:

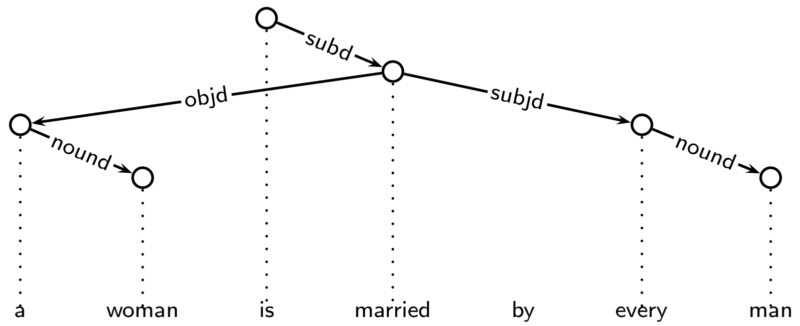
$$\textit{married} \quad = \quad \left[\begin{array}{lll} \text{link} & : & \left[\begin{array}{lll} \text{arg1} & : & \{\text{subjd}\} \\ \text{arg2} & : & \{\text{objd}\} \end{array} \right] \end{array} \right]$$

- remedied direct linking principle:
  $v \xrightarrow{l}_{\text{PA}} v' \ \wedge \ l' \in \text{link}(v)(l) \quad \Rightarrow \quad v \xrightarrow{l'}_{\text{DS}} v'$

Indirect linking principle example

- example ID tree and DS dag:





- indirect linking ensures that the surface subject of the passive auxiliary realizes a deep object:

$$is \stackrel{\text{subj}}{\longrightarrow}_{\text{ID}} a \quad \Rightarrow \quad is \stackrel{\text{subd}}{\longrightarrow} \longrightarrow^{*} \stackrel{\text{objd}}{\longrightarrow}_{\text{DS}} a$$

Indirect linking principle

- lexical entry for passive auxiliary *is*:

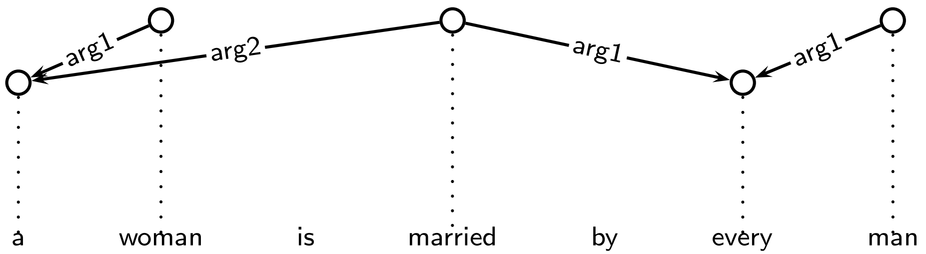
$$is \quad = \quad \left[\ \mathsf{ilink} \quad : \quad \left[\ \mathsf{subj} \quad : \quad (\mathsf{subd}, \mathsf{objd})\ \right]\ \right]$$

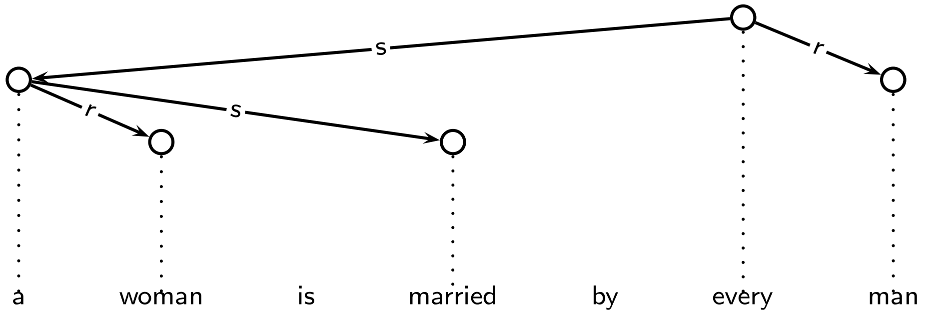
- indirect linking principle:

$$v \xrightarrow{l}_{\mathrm{ID}} v' \ \wedge \ (l', l'') \in \mathsf{ilink}(v)(l) \quad \Rightarrow \quad v \xrightarrow{l'} \rightarrow^{*} \xrightarrow{l''}_{\mathrm{DS}} v'$$

Contra-dominance principle example

- example PA dag and SC tree:





- contra-dominance ensures that verbs get into the scope of their quantifier arguments:

$$married \xrightarrow{\text{arg1}}_{\text{PA}} every \quad \Rightarrow \quad every \xrightarrow{\text{s}} \rightarrow^{*}_{\text{SC}} married$$

Contra-dominance principle

- lexical entry for transitive verb *married*:

$$\textit{married} \quad = \quad \left[\; \mathsf{contradom} \quad : \quad \left[\begin{array}{lll} \mathsf{arg1} & : & \{\mathsf{s}\} \\ \mathsf{arg2} & : & \{\mathsf{s}\} \end{array} \right] \right]$$

- contra-dominance principle:
$v \xrightarrow{l}_{\mathrm{PA}} v' \; \wedge \; l' \in \mathsf{contradom}(v)(l) \quad \Rightarrow \quad v' \xrightarrow{l'} \rightarrow^{*}_{\mathrm{SC}} v$

XDG parser

- actually, a constraint solver which can be used also for parsing
- XDG parsing encoded as a constraint satisfaction problem on finite sets of integers (Duchier MOL 1999), in Mozart-Oz (`www.mozart-oz.org`)
- concurrent: all dimensions processed in parallel
- worst-case complexity: NP-complete
- average-case complexity: highly grammar-dependent, polynomial for small test grammars, parsing of large induced grammars (XTAG, PDT induced) not yet feasible
- ongoing research: what is the precise parsing complexity of any given XDG grammar, when can we expect a grammar to be well-behaved?

XDG parser system

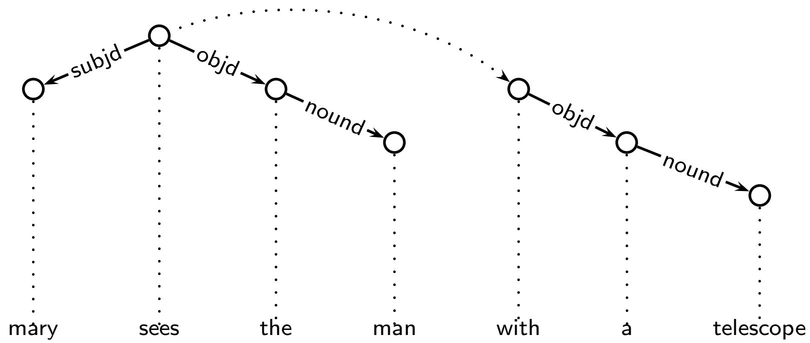
- lots of nice features (GUI, grammar type checking, different grammar languages, tools for evaluation, XML support)
- extensive documentation
- easy to install and use
- runs on MacOS X, Unix and Windows

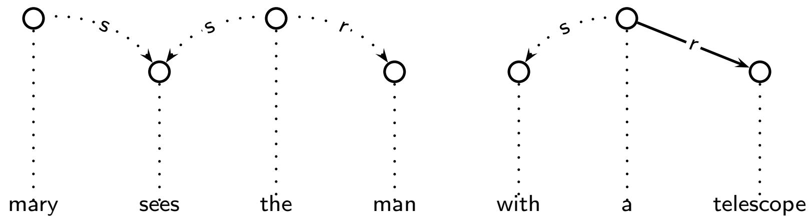
Underspecifi cation and preferences

- XDG parser allows us to postpone the enumeration of solutions on each dimension individually
- before continuing search, we can always reflect the current partial parse including all the information obtained so far on all dimensions
- using this information, we can guide search e.g. by preferences
- preferences architecture: work in progress, first published in (Dienes, Koller and Kuhlmann Nancy 2003)

Preferences example

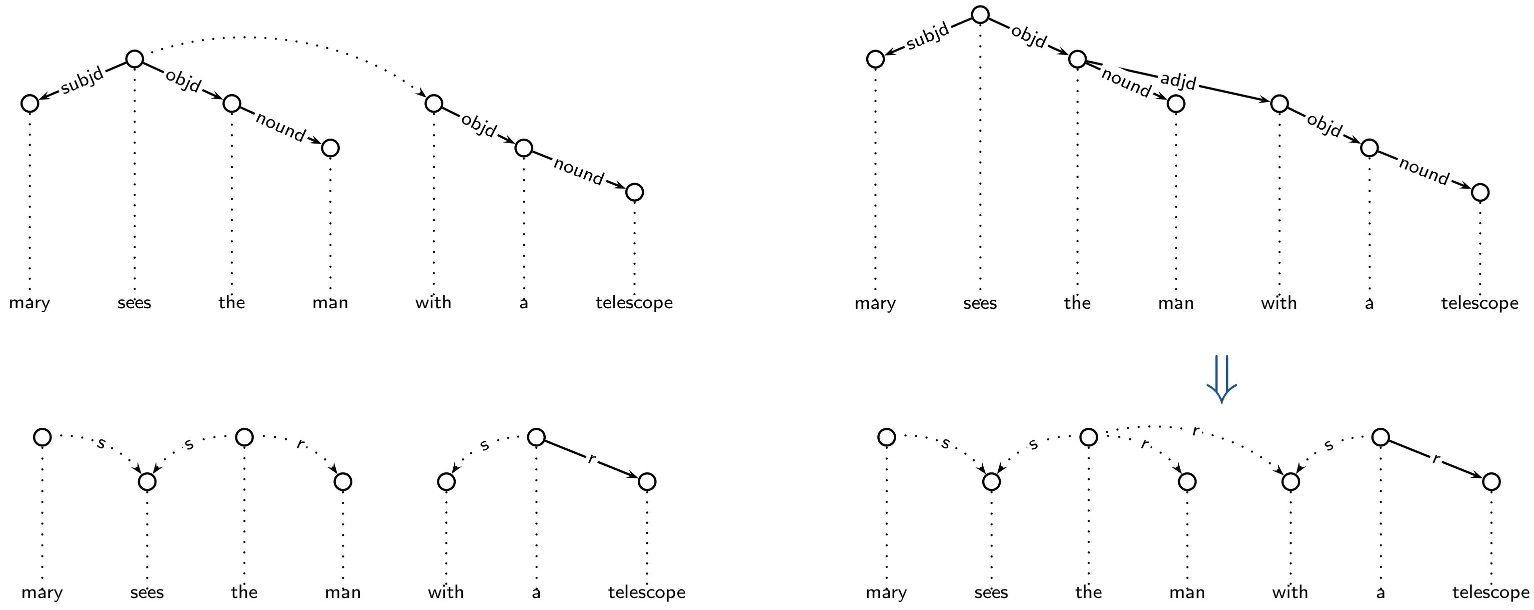
- new example: *Mary sees the man with a telescope.*
- underspecified DS dag and SC tree:





Inference from syntax to semantics

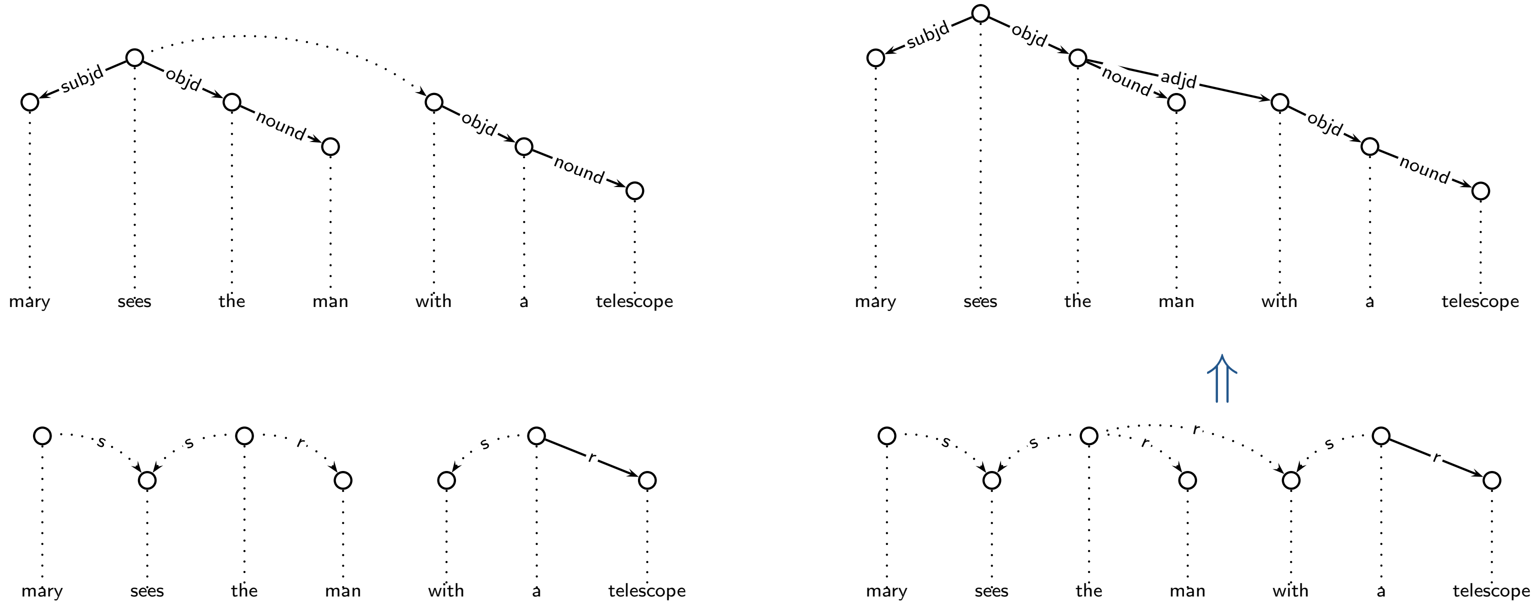
- assuming that we have a preference component that tells us that the PP is more likely to attach to the NP...
- ... then, this *syntactic* inference entails the *semantic* inference that the PP gets into the restriction of the NP:



- inference (co-dominance): $the \overset{\text{adjd}}{\longrightarrow}_{\mathrm{DS}} with \quad \Rightarrow \quad the \overset{\text{r}}{\longrightarrow} \longrightarrow^{*}_{\mathrm{SC}} with$

Inference from semantics to syntax

- assuming that we have a preference component that tells us that *with* is in the restriction of the NP...
- ... then, this *semantic* inference entails the *syntactic* inference that the PP modifies the NP:



- inference (falsified contra-dominance):

$$sees \xrightarrow{\text{advd}}_{\mathrm{DS}} with \quad \Rightarrow \quad with \xrightarrow{\text{s}} \rightarrow^{*}_{\mathrm{SC}} sees$$

Conclusion

- introduced XDG meta grammar formalism
- represents both syntactic and semantic dimensions in the same system
- correspondence between syntax and semantics relational instead of functional
- allows concurrent processing of syntax and semantics
- preferences can trigger inferences into any direction
- linguistic dimensions become more autonomous
- grammar development becomes more modular

Outlook

- deeper understanding of XDG and XDG parsing
  - what precisely are possible principles?
  - what is the precise parsing complexity of any given XDG grammar, when can we expect a grammar to be well-behaved?
- continue work on the preference architecture (started in Dienes, Koller, and Kuhlmann Nancy 2003)
- start writing my dissertation :-)

Demo

- anyone interested in a demo?