DG(X) - eXtensible Dependency Grammar

Ralph Debusmann

Joint work with Joachim Niehren and Denys Duchier

Programming Systems Lab

Universität des Saarlandes

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)
- motivation: handling free word order languages

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)
- motivation: handling free word order languages
- most work done on German, using the traditional theory of topological fields (19th century)

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)
- motivation: handling free word order languages
- most work done on German, using the traditional theory of topological fields (19th century)
- efficient parsing by constraint programming

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)
- motivation: handling free word order languages
- most work done on German, using the traditional theory of topological fields (19th century)
- efficient parsing by constraint programming
- declarative grammar formalism: Duchier/Debusmann, ACL 2001

Topological Dependency Grammar (TDG)

- introduced by Denys Duchier (MOL, 1999)
- motivation: handling free word order languages
- most work done on German, using the traditional theory of topological fields (19th century)
- efficient parsing by constraint programming
- declarative grammar formalism: Duchier/Debusmann, ACL 2001
- only handles syntax (no syntax-semantics interface)

General Goals

- go all the way from syntax to semantics

General Goals

- go all the way from syntax to semantics
- do compositional semantics construction (Montague 69)

General Goals

- go all the way from syntax to semantics
- do compositional semantics construction (Montague 69)
- semantics formalism shall support underspecification (Hobbs 88, Reyle 93, Pinkal 96, etc.)

General Goals

- go all the way from syntax to semantics
- do compositional semantics construction (Montague 69)
- semantics formalism shall support underspecification (Hobbs 88, Reyle 93, Pinkal 96, etc.)
- draw early inferences

General Goals

- go all the way from syntax to semantics
- do compositional semantics construction (Montague 69)
- semantics formalism shall support underspecification (Hobbs 88, Reyle 93, Pinkal 96, etc.)
- draw early inferences
- use preference information

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs
- introduced by Debusmann/Duchier 2003

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs
- introduced by Debusmann/Duchier 2003
- parameter X defines types and properties of the graphs

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs
- introduced by Debusmann/Duchier 2003
- parameter X defines types and properties of the graphs
- each graph is viewed as a separate dimension: e.g. syntactic tree, topology tree, semantic dag (directed acyclic graph)

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs
- introduced by Debusmann/Duchier 2003
- parameter X defines types and properties of the graphs
- each graph is viewed as a separate dimension: e.g. syntactic tree, topology tree, semantic dag (directed acyclic graph)
- Topological Dependency Grammar: TDG = DG(d,t)

eXtensible Dependency Grammar DG(X)

- new description language for sets of graphs
- introduced by Debusmann/Duchier 2003
- parameter X defines types and properties of the graphs
- each graph is viewed as a separate dimension: e.g. syntactic tree, topology tree, semantic dag (directed acyclic graph)
- Topological Dependency Grammar: TDG = DG(d,t)
- compositional semantics construction: DG(d,t,s)

Overview

1. Topological Dependency Grammar: TDG
2. eXtensible Dependency Grammar: DG(X)
3. TDG = DG(d,t)
4. Semantics construction: DG(d,t,s)
5. System demo
6. Summary and outlook

Dependency grammar formalisms

- Meaning Text Theory (MTT) (Melcuk 88)

Dependency grammar formalisms

- Meaning Text Theory (MTT) (Melcuk 88)
- Functional Generative Description (FGD) (Sgall et al 86)

Dependency grammar formalisms

- Meaning Text Theory (MTT) (Melcuk 88)
- Functional Generative Description (FGD) (Sgall et al 86)
- Topological Dependency Grammar (TDG) (Duchier 99)

Dependency grammar formalisms

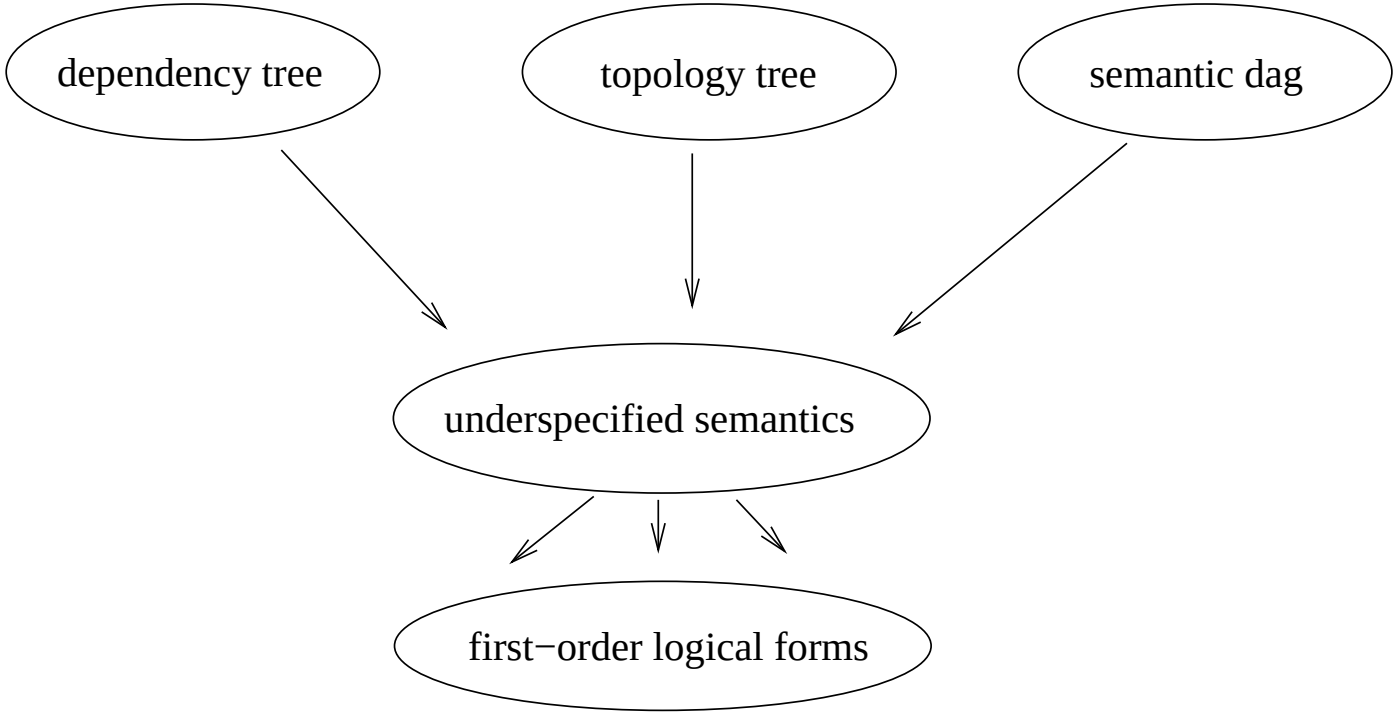
- Meaning Text Theory (MTT) (Melcuk 88)
- Functional Generative Description (FGD) (Sgall et al 86)
- Topological Dependency Grammar (TDG) (Duchier 99)
- Free Order Dependency Grammar (FODG) (Holan/Kubon/Platek 2001)

Our take on dependency grammar

dimensions of analysis:

Our take on dependency grammar

dimensions of analysis:



Toplogical Dependency Grammar (TDG)

- developed to handle relatively free word order in German

Toplogical Dependency Grammar (TDG)

- developed to handle relatively free word order in German
- makes use of the notion of topological fields

Toplogical Dependency Grammar (TDG)

- developed to handle relatively free word order in German
- makes use of the notion of topological fields
- two dimensions: dependency tree (d) and topology tree (t)

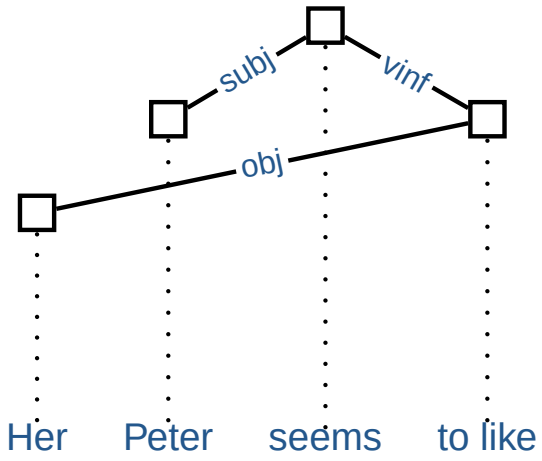
Toplogical Dependency Grammar (TDG)

- developed to handle relatively free word order in German
- makes use of the notion of topological fields
- two dimensions: dependency tree (d) and topology tree (t)
- dependency tree unordered, topology tree ordered and projective

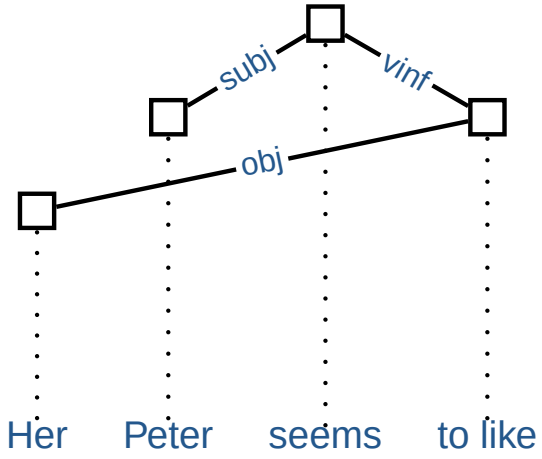
Toplogical Dependency Grammar (TDG)

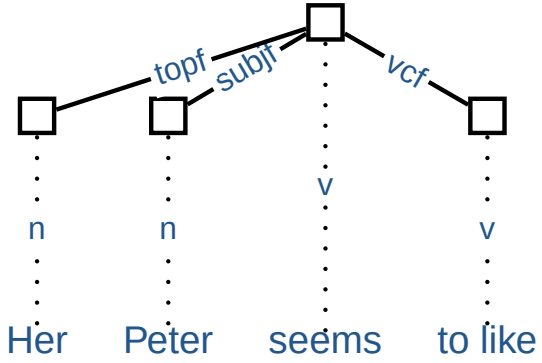
- developed to handle relatively free word order in German
- makes use of the notion of topological fields
- two dimensions: dependency tree (d) and topology tree (t)
- dependency tree unordered, topology tree ordered and projective
- topology tree is a flattening of the dependency tree

Example

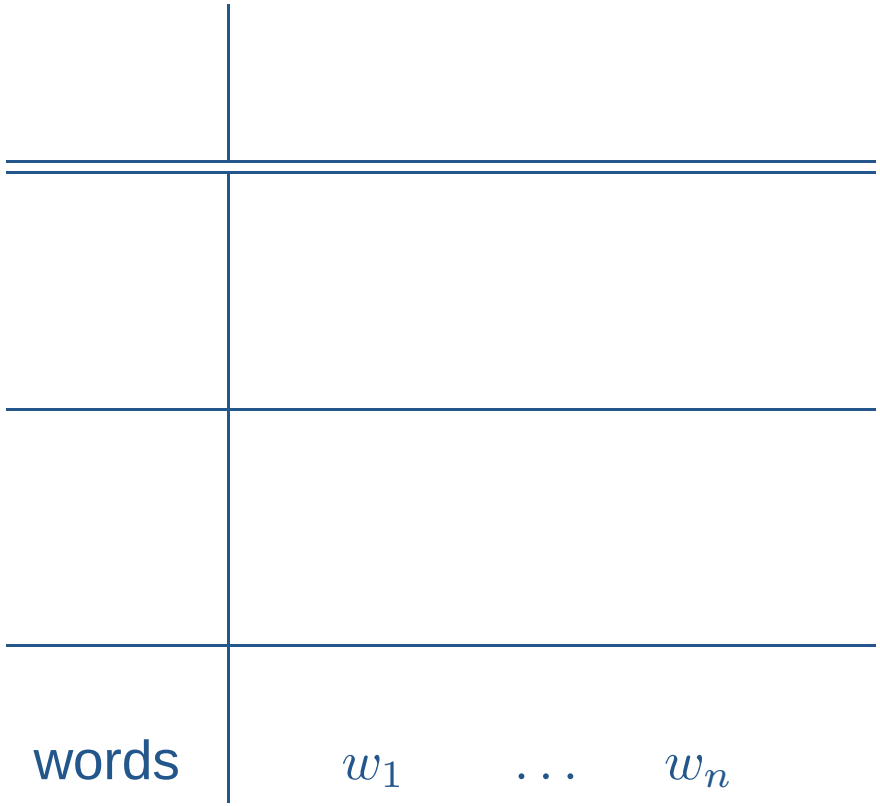


Example


subj
vinf
obj
Her
Peter
seems
to like

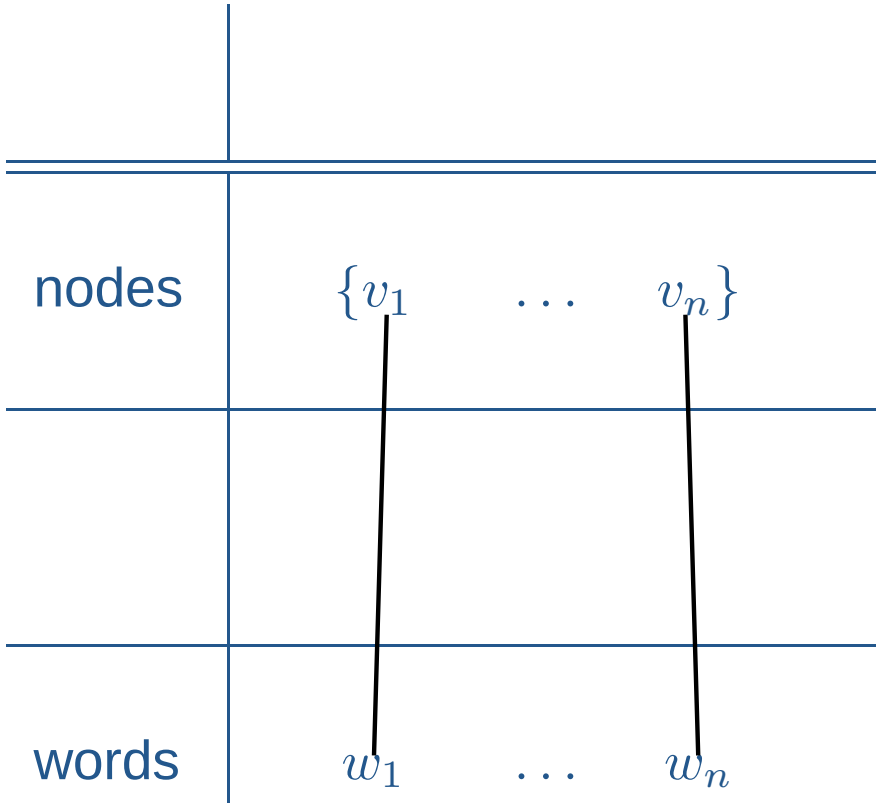

topf
subjf
vcf
n
n
v
v
Her
Peter
seems
to like

XDG architecture



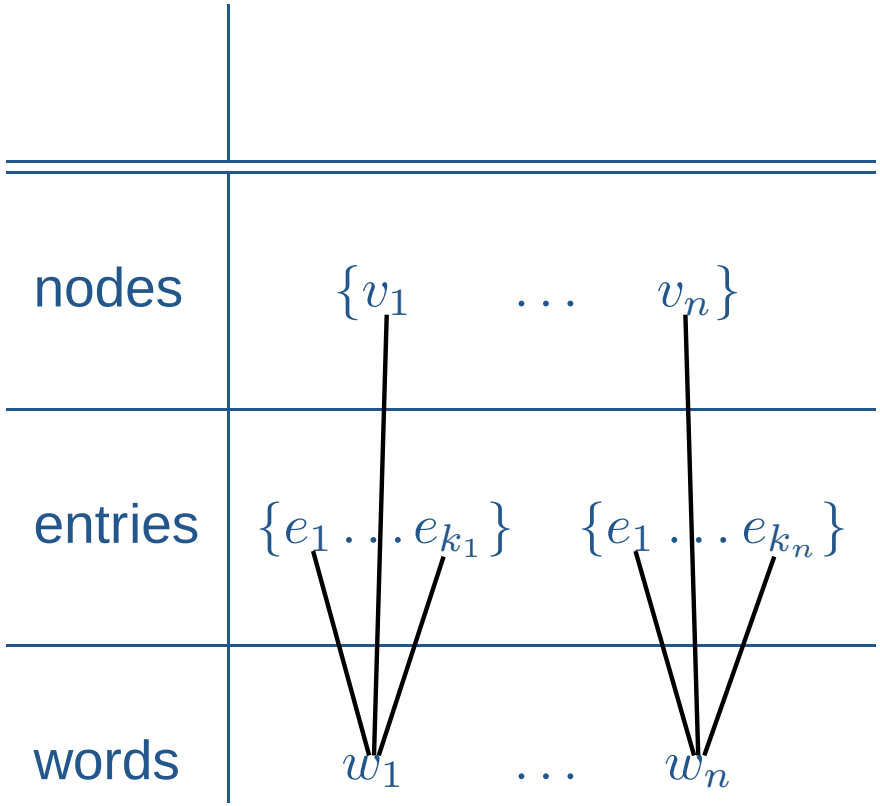
- $w_1 \ldots w_n$ is the input string

XDG architecture



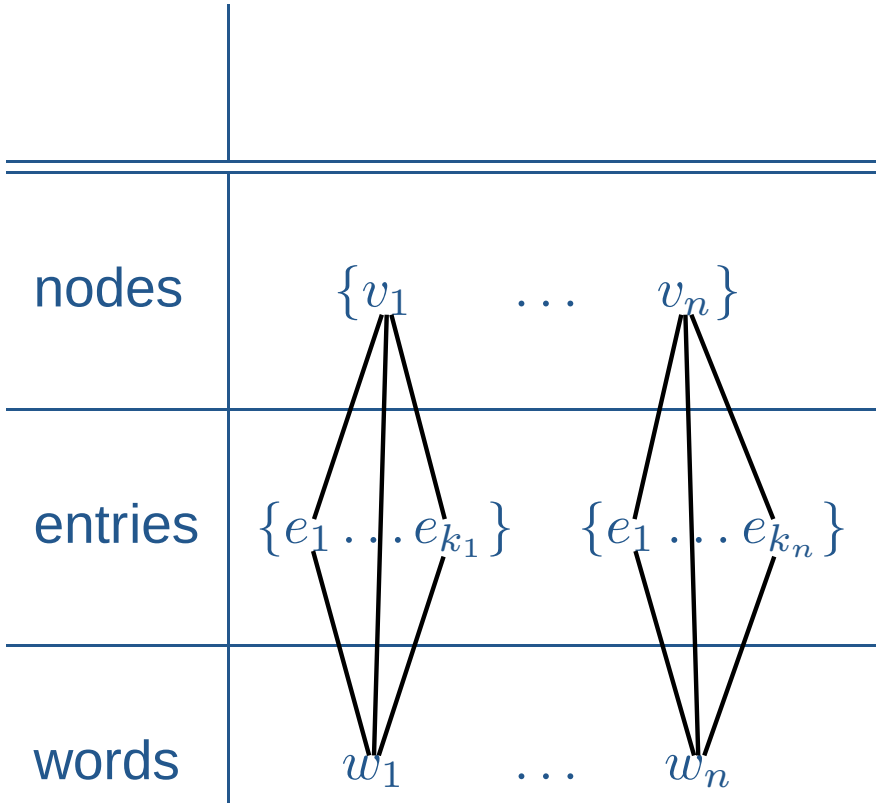
- each word w_i corresponds one-to-one to nodes v_i in node set V

XDG architecture



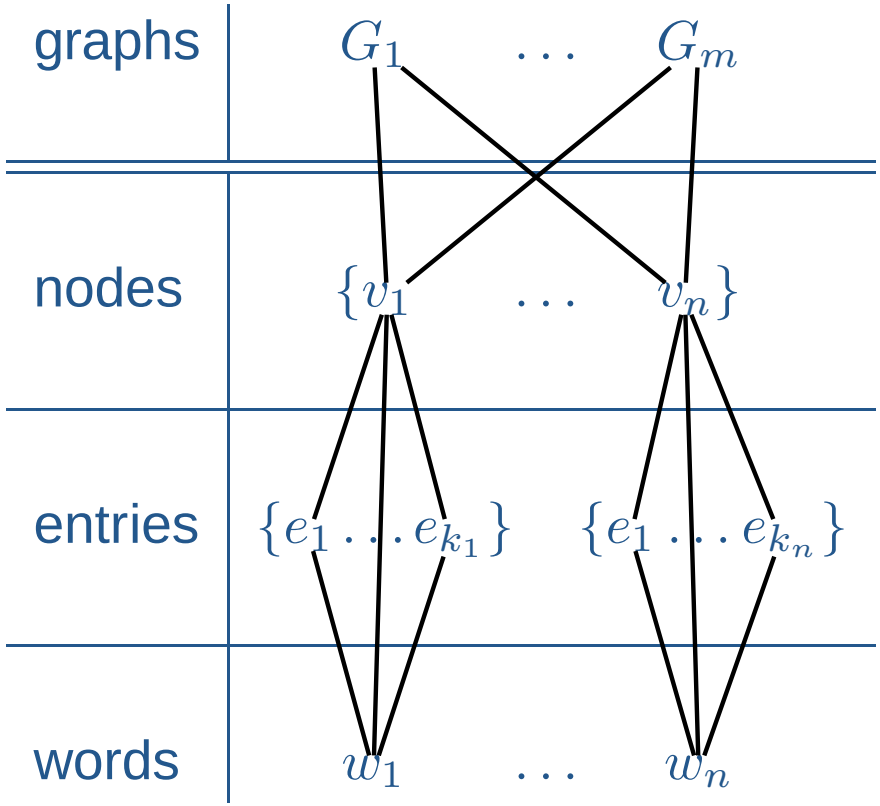
- each word w_i in the input string is assigned a set of lexical entries $\{e_1, \ldots, e_{k_i}\}$

XDG architecture



- each analysis selects for each node one of these lexical entries

XDG architecture



- the same node set V is shared across the m graphs $G_1, \ldots, G_m$

Graph G

V shared set of nodes

Graph G

V	shared set of nodes
L	set of labels for G

Graph G

V	shared set of nodes
L	set of labels for G
$V \times L \times V$	labeled edges

Graph G

V	shared set of nodes
L	set of labels for G
$V \times L \times V$	labeled edges
E	set of lexical entries

Graph G

V	shared set of nodes
L	set of labels for G
$V \times L \times V$	labeled edges
E	set of lexical entries
lex : $V \rightarrow E$	lexical entry assignment

Principles

- intra-dimensional: $P(G)$

Principles

- intra-dimensional: $P(G)$
- inter-dimensional: $P(G_1, G_2)$

Principles

- intra-dimensional: $P(G)$
- inter-dimensional: $P(G_1, G_2)$
- predefined principles library available

Principles

- intra-dimensional: $P(G)$
- inter-dimensional: $P(G_1, G_2)$
- predefined principles library available
- more principles by constraint programming

Dag principle

$\mathsf{dag}(G)$: G is a directed acyclic graph.

Tree principle

$\mathsf{tree}(G)$: G is a tree.

In and out principles

- modalities (zero, one, zero or one, any number):

$$M = \{0, 1, ?, *\}$$

In and out principles

- modalities (zero, one, zero or one, any number):

$$M = \{0, 1, ?, *\}$$

- lexicalized functions: in and out of type $V \rightarrow (L \rightarrow M)$

In and out principles

- modalities (zero, one, zero or one, any number):

$$M = \{0, 1, ?, *\}$$

- lexicalized functions: in and out of type $V \rightarrow (L \rightarrow M)$
- $\mathsf{in}(G, \mathsf{in})$

In and out principles

- modalities (zero, one, zero or one, any number):

$$M = \{0, 1, ?, *\}$$

- lexicalized functions: in and out of type $V \rightarrow (L \rightarrow M)$
- $\mathsf{in}(G, \mathsf{in})$
- $\mathsf{out}(G, \mathsf{out})$

Order principle

- lexicalized function lab : $V \rightarrow L$ assigns node labels to nodes

Order principle

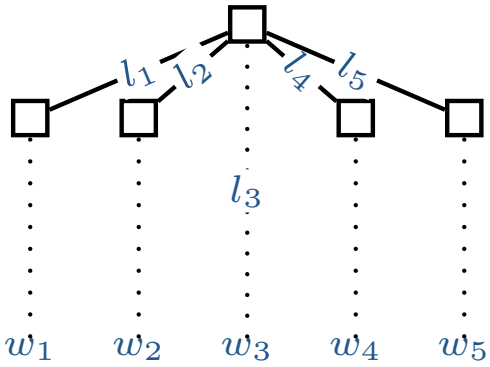
- lexicalized function $\mathsf{lab} : V \rightarrow L$ assigns node labels to nodes
- total order $\prec$ on labels

Order principle

- lexicalized function $\mathsf{lab} : V \rightarrow L$ assigns node labels to nodes
- total order $\prec$ on labels
- $\mathsf{order}(G, \mathsf{lab}, \prec)$

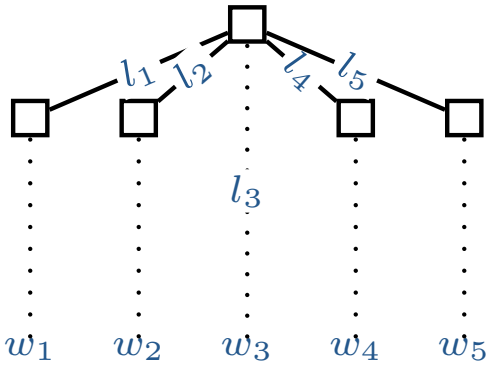
Order principle

- lexicalized function lab : $V \rightarrow L$ assigns node labels to nodes
- total order $\prec$ on labels
- order$(G, \text{lab}, \prec)$
- example:



Order principle

- lexicalized function lab : $V \rightarrow L$ assigns node labels to nodes
- total order $\prec$ on labels
- order$(G, \text{lab}, \prec)$
- example:



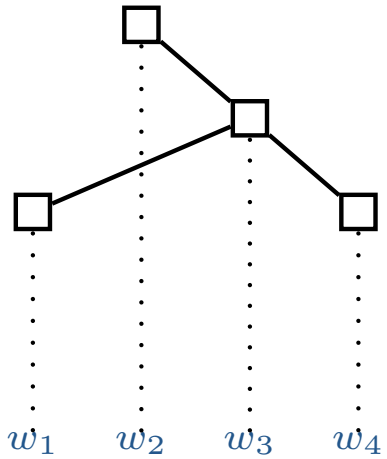
- $$l_i \prec l_j \quad \Rightarrow \quad w_i < w_j$$

Flattening principle

flatten(G_1, G_2): G_2 is a flatter dag than G_1

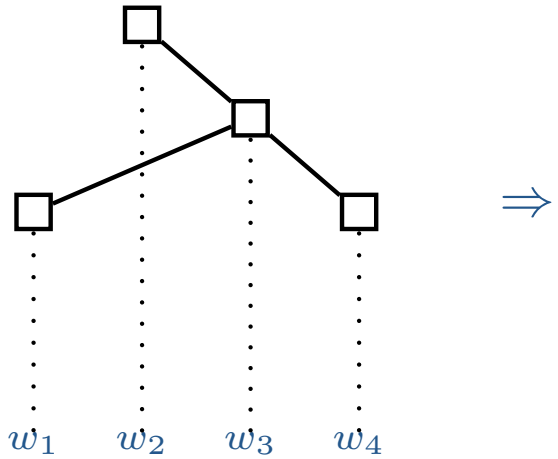
Flattening principle

flatten(G_1, G_2): G_2 is a flatter dag than G_1



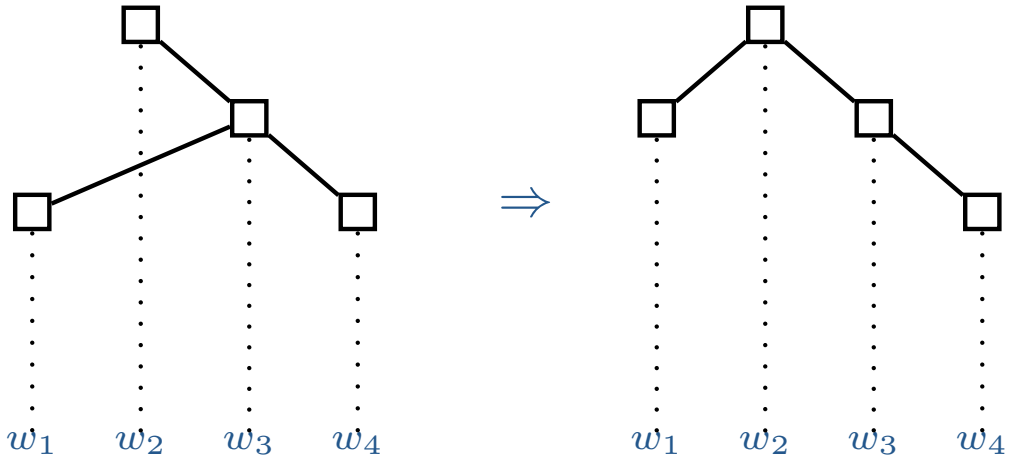
Flattening principle

flatten(G_1, G_2): G_2 is a flatter dag than G_1



Flattening principle

flatten(G_1, G_2): G_2 is a flatter dag than G_1



Topological Dependency Grammar: TDG = DG(d,t)

- Dependency tree (d):

Topological Dependency Grammar: TDG = DG(d,t)

- Dependency tree (d):

$$L_d \quad = \quad \{\mathsf{subj}, \mathsf{obj}, \mathsf{vinf}, \ldots\}$$

Topological Dependency Grammar: TDG = DG(d,t)

- Dependency tree (d):

$$L_d \quad = \quad \{\mathsf{subj}, \mathsf{obj}, \mathsf{vinf}, \ldots\}$$

- Topology tree (t):

Topological Dependency Grammar: TDG = DG(d,t)

- Dependency tree (d):

$$L_d \quad = \quad \{\mathsf{subj}, \mathsf{obj}, \mathsf{vinf}, \ldots\}$$

- Topology tree (t):

$$L_t \quad = \quad \{\mathsf{topf}, \mathsf{subjf}, \mathsf{vcf}, \ldots\}$$

Lexicon

- lexical entry signature:

$$\left[\begin{array}{lcl} d & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_d \to M \\ \mathsf{out} & : & L_d \to M \end{array}\right] \\ t & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_t \to M \\ \mathsf{out} & : & L_t \to M \\ \mathsf{lab} & : & L_t \end{array}\right] \end{array}\right]$$

Lexicon

- lexical entry signature:

$$\left[\begin{array}{lcl} d & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_d \to M \\ \mathsf{out} & : & L_d \to M \end{array}\right] \\ t & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_t \to M \\ \mathsf{out} & : & L_t \to M \\ \mathsf{lab} & : & L_t \end{array}\right] \end{array}\right]$$

- in, out and lab-functions lexicalized i.e. defined through the lexicon

Lexicon

- lexical entry signature:

$$\left[\begin{array}{lcl} d & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_d \to M \\ \mathsf{out} & : & L_d \to M \end{array}\right] \\ t & : & \left[\begin{array}{rcl} \mathsf{in} & : & L_t \to M \\ \mathsf{out} & : & L_t \to M \\ \mathsf{lab} & : & L_t \end{array}\right] \end{array}\right]$$

- in, out and lab-functions lexicalized i.e. defined through the lexicon
- e.g. in_d:

$$\mathsf{in}_d(v) \quad = \quad \mathsf{lex}(v).d.\mathsf{in}$$

Principles

dependency	topology
$\mathsf{tree}(d)$	$\mathsf{tree}(t)$
$\mathsf{in}(d, \mathsf{in}_d)$	$\mathsf{in}(t, \mathsf{in}_t)$
$\mathsf{out}(d, \mathsf{out}_d)$	$\mathsf{out}(t, \mathsf{in}_t)$
	$\mathsf{order}(t, \mathsf{lab}_t, \mathsf{topf} \prec \mathsf{subjf} \prec \mathsf{vcf})$
$\mathsf{flatten}(d, t)$	

Semantics construction: DG(d,t,s)

- additional semantic dag (s)

Semantics construction: DG(d,t,s)

- additional semantic dag (s)
- labels e.g. act (actor) and pat (patient)

Semantics construction: DG(d,t,s)

- additional semantic dag (s)
- labels e.g. act (actor) and pat (patient)
- underspecified semantic representation computed from dependency tree and semantic dag

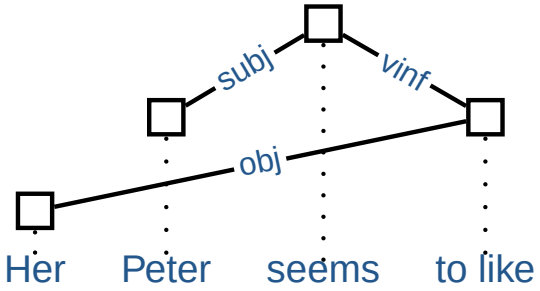
Semantics construction: DG(d,t,s)

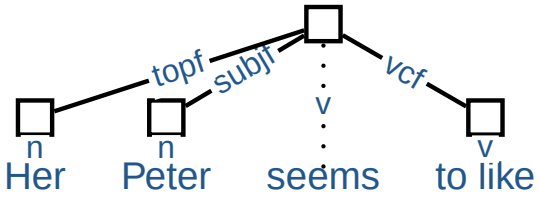
- additional semantic dag (s)
- labels e.g. act (actor) and pat (patient)
- underspecified semantic representation computed from dependency tree and semantic dag
- underspecification formalism: Constraint Language for Lambda Structures (CLLS) (Niehren et al. 98)

Semantics construction: DG(d,t,s)

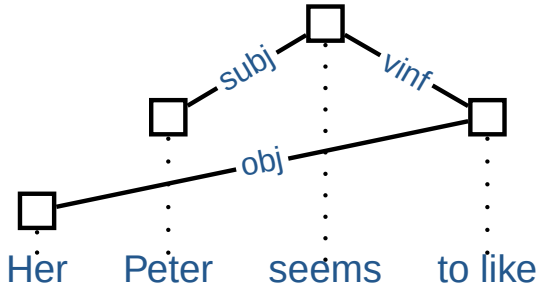
- additional semantic dag (s)
- labels e.g. act (actor) and pat (patient)
- underspecified semantic representation computed from dependency tree and semantic dag
- underspecification formalism: Constraint Language for Lambda Structures (CLLS) (Niehren et al. 98)
- CLLS based on dominance constraints (Marcus/Hindle/Fleck 83)

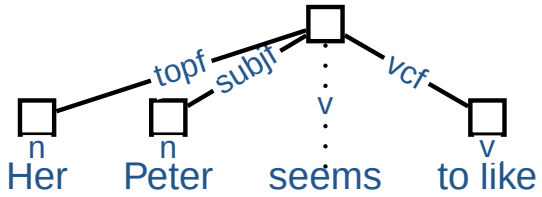
Example

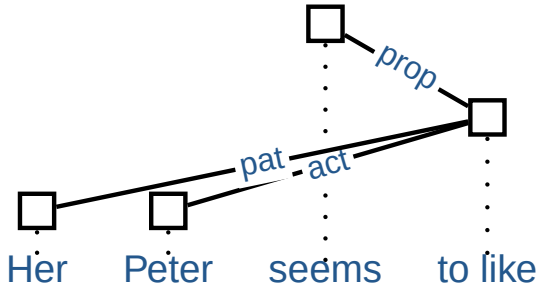

subj
vinf
obj
Her
Peter
seems
to like


topf
subjf
vcf
v
n
n
v
Her
Peter
seems
to like

Example


subj
vinf
obj
Her
Peter
seems
to like


topf
subjf
vcf
v
n
n
v
Her
Peter
seems
to like


prop
pat
act
Her
Peter
seems
to like

Principles for the semantic dag

- dag(s)

Principles for the semantic dag

- dag(s)
- flatten(s,d)

Principles for the semantic dag

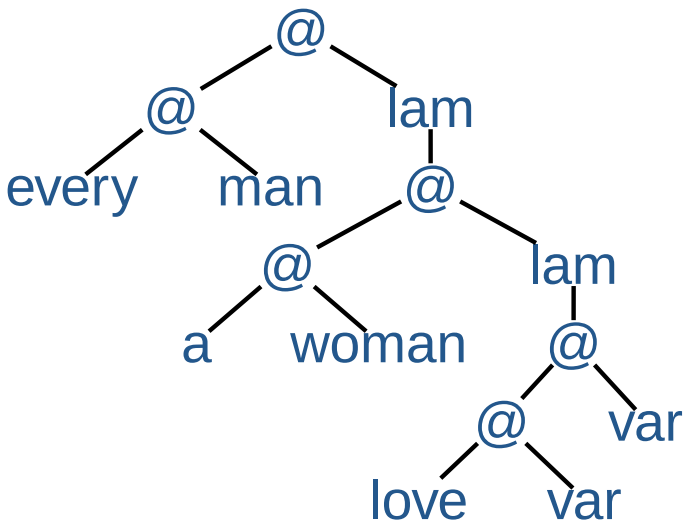
- dag(s)
- flatten(s,d)
- lexicalized linking principle mapping e.g. actor to subject

CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):

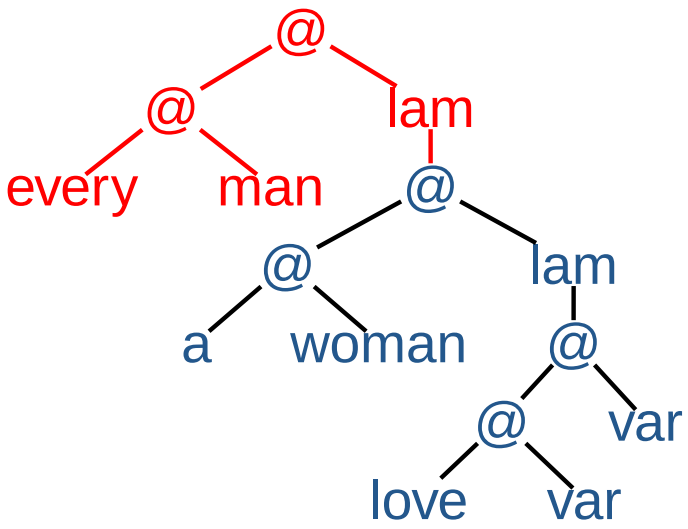
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



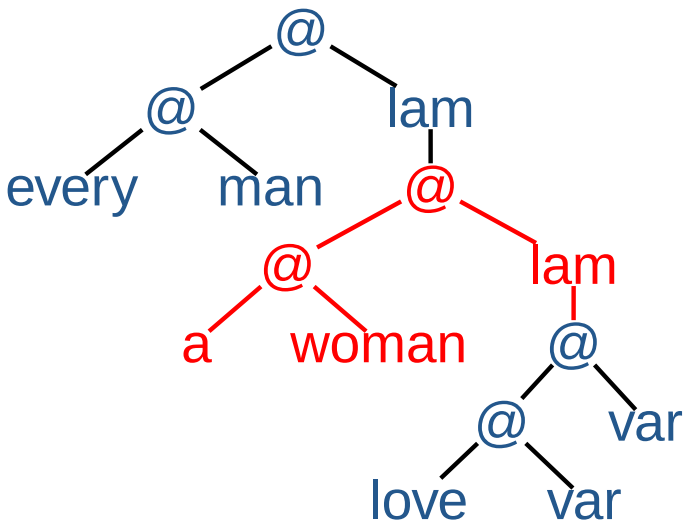
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



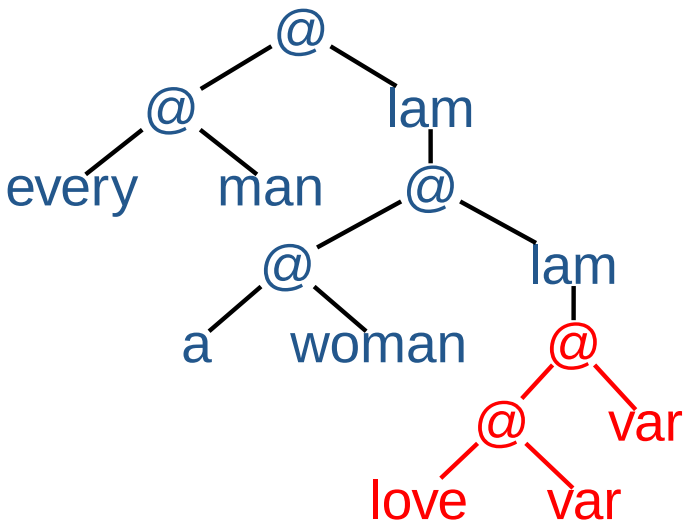
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



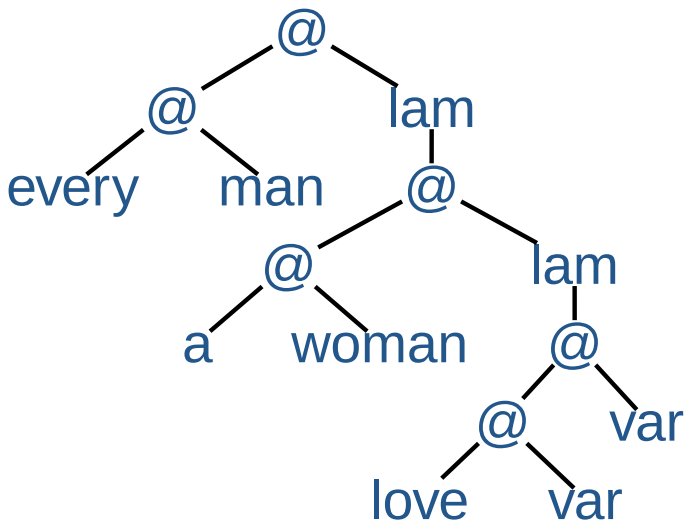
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



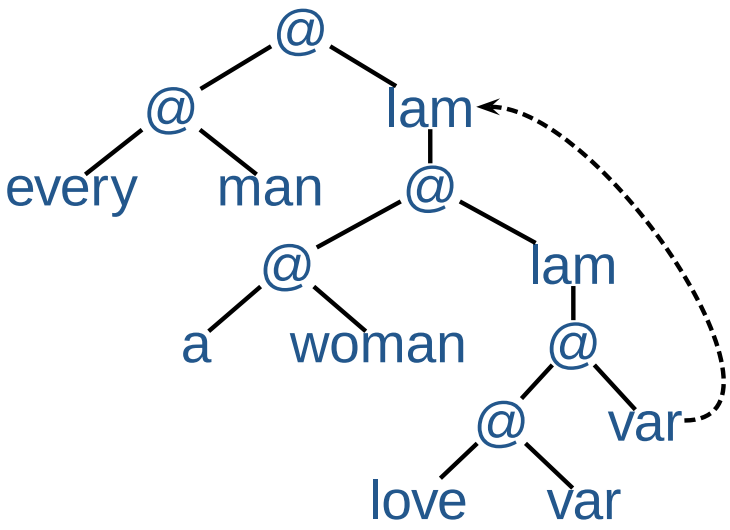
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



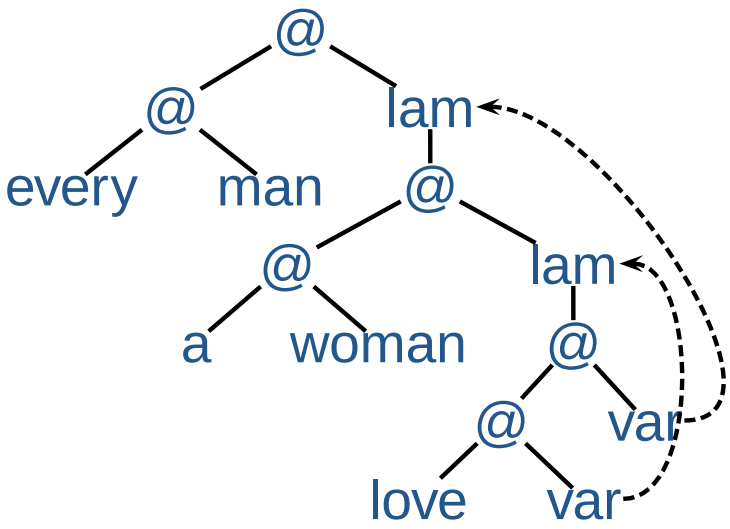
CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):



CLLS semantics representation

- CLLS lambda structure for “Every man loves a woman” (weak reading):

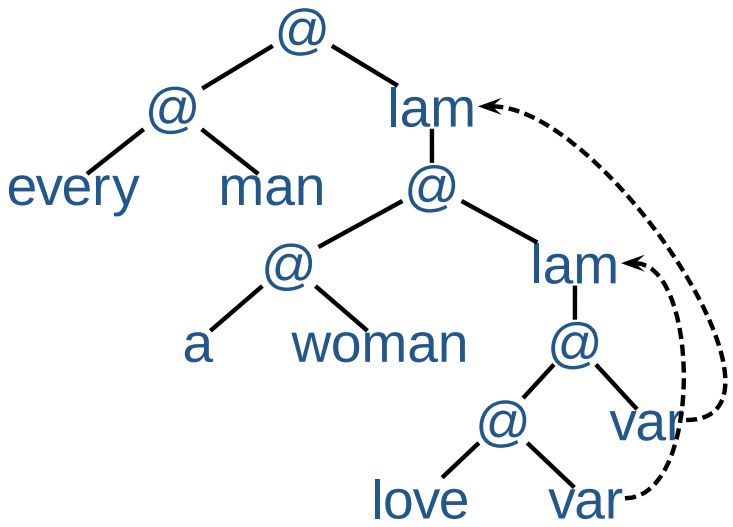


CLLS semantics representation

- both readings (also the strong one):

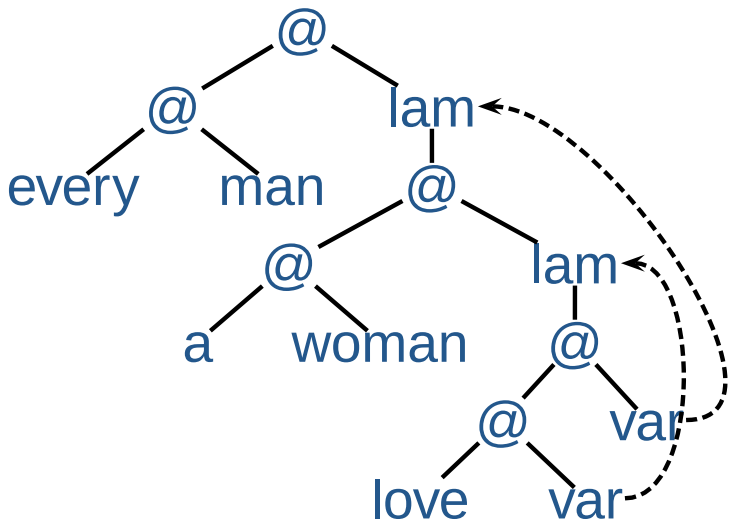
CLLS semantics representation

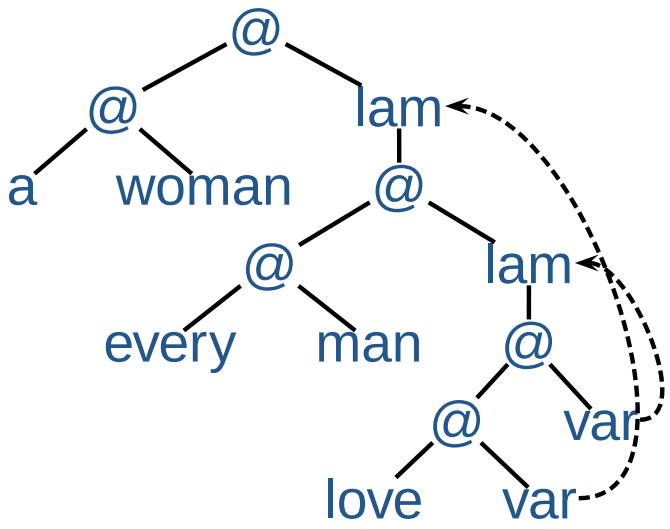
- both readings (also the strong one):



CLLS semantics representation

- both readings (also the strong one):



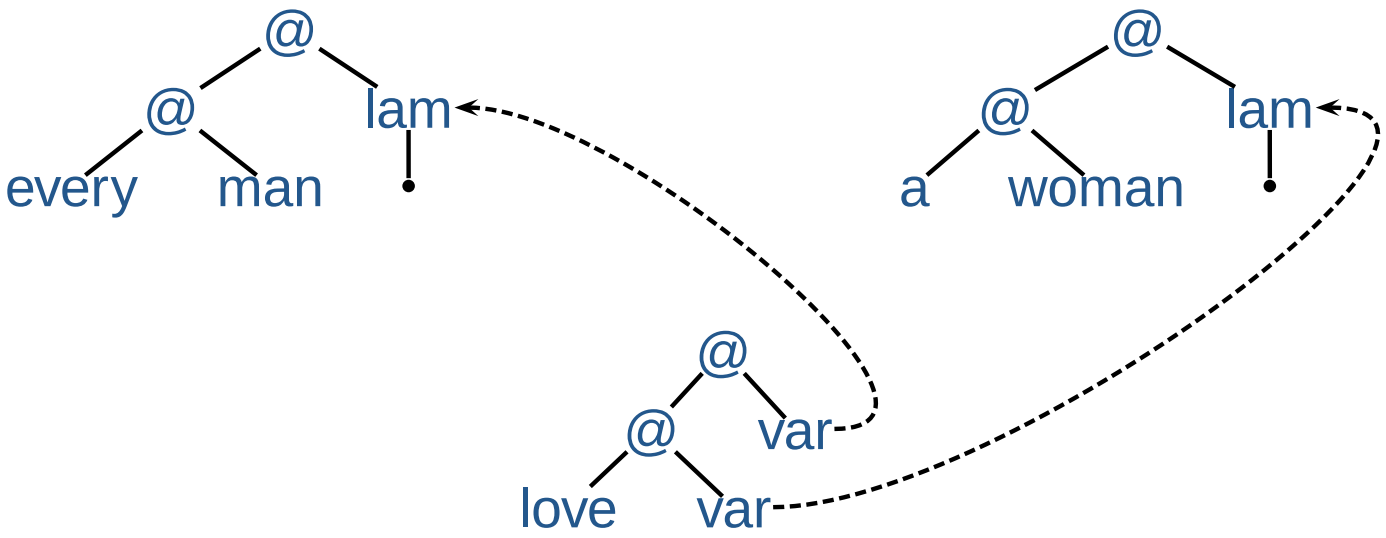


Underspecifi ed semantics

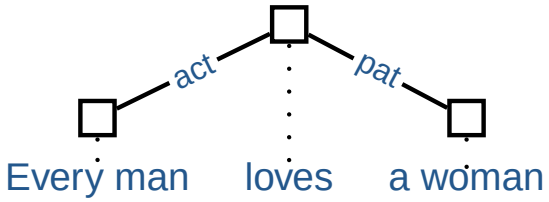
- underspecified representation of both readings:

Underspecifi ed semantics

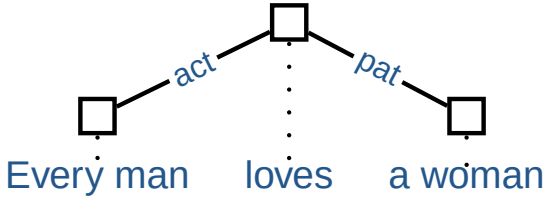
- underspecified representation of both readings:

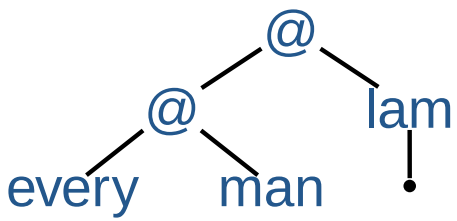


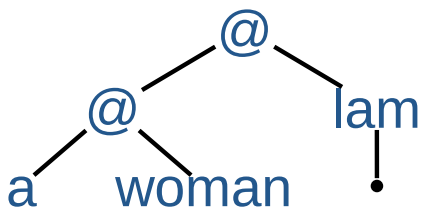
Semantics construction

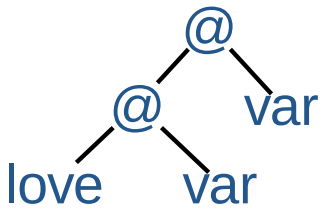


Semantics construction

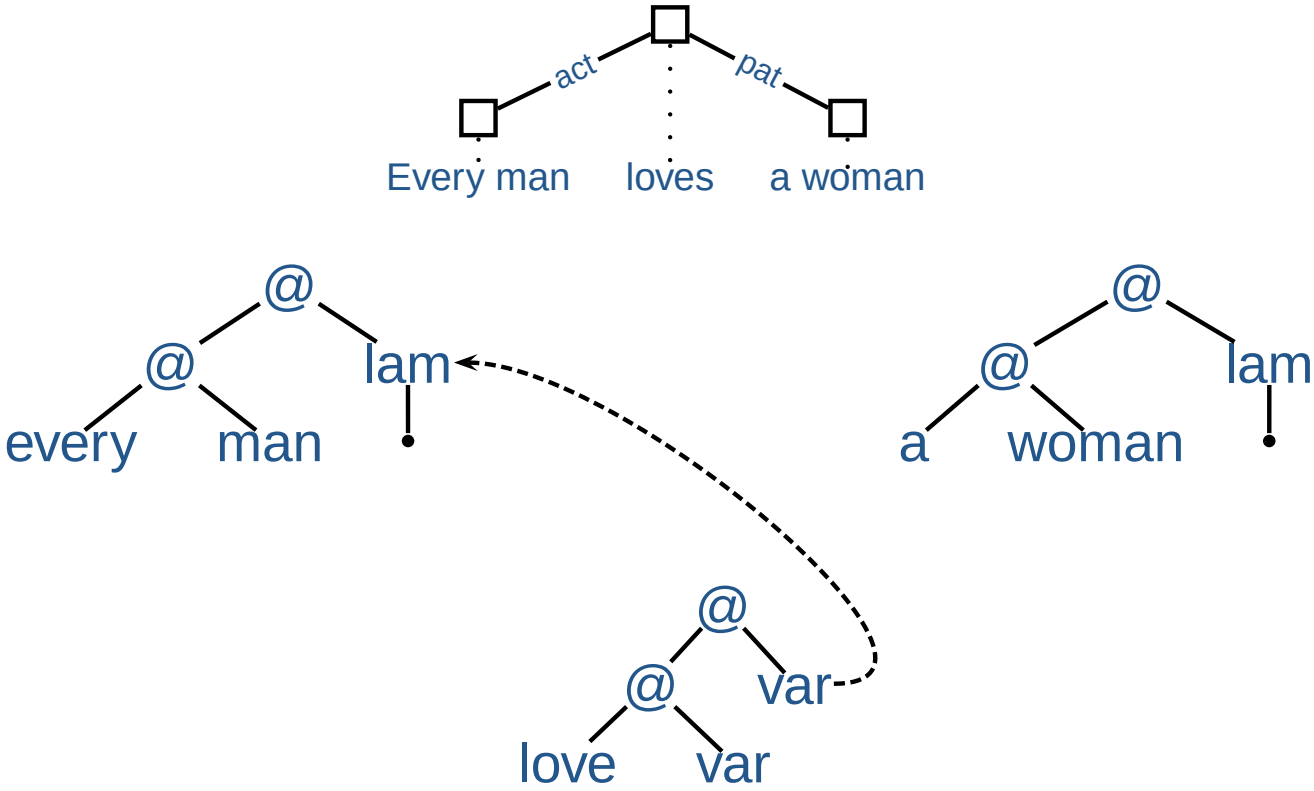




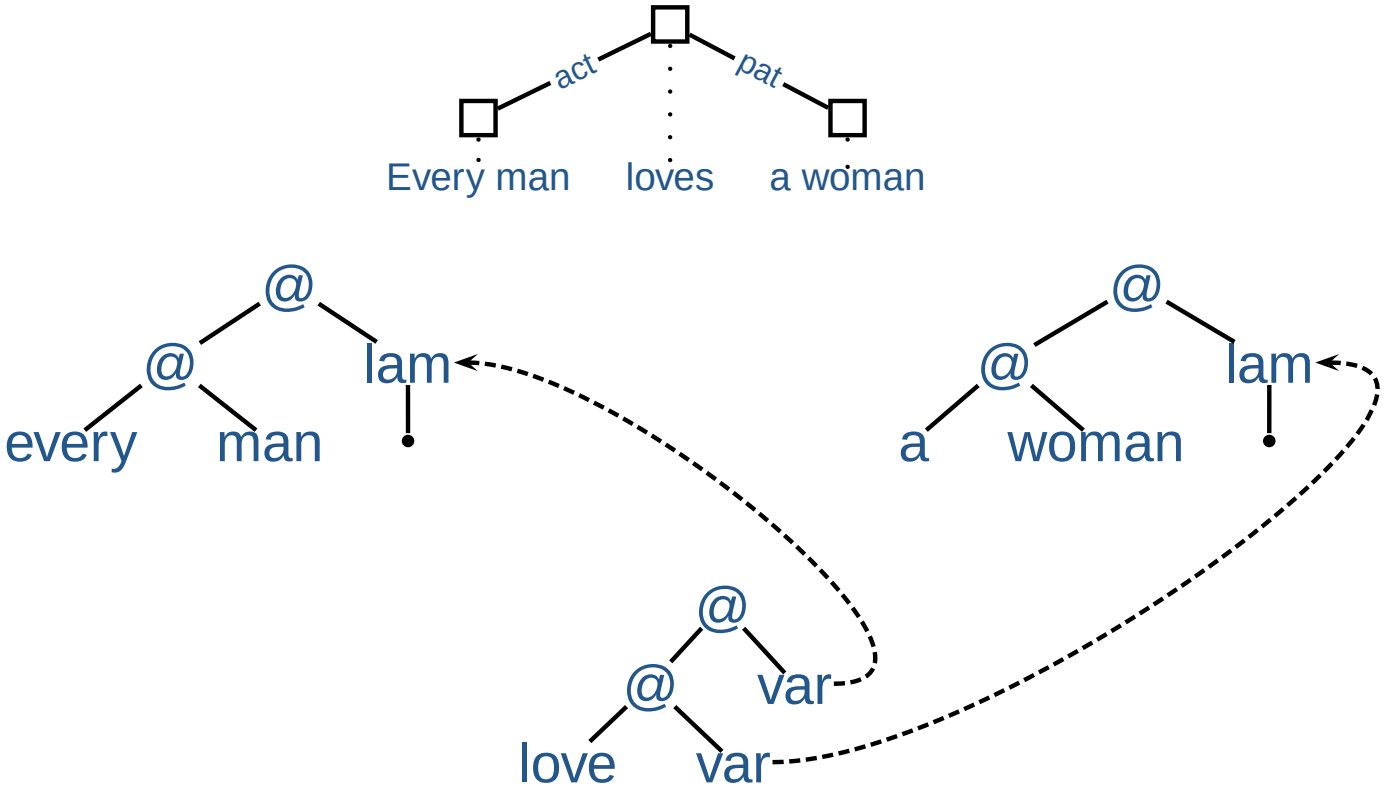




Semantics construction



Semantics construction



System demo

Summary

- introduced eXtensible Dependency Grammar DG(X)

Summary

- introduced eXtensible Dependency Grammar DG(X)
- DG(X): graph description language

Summary

- introduced eXtensible Dependency Grammar DG(X)
- DG(X): graph description language
- TDG = DG(d,t)

Summary

- introduced eXtensible Dependency Grammar DG(X)
- DG(X): graph description language
- TDG = DG(d,t)
- compositional semantics composition: DG(d,t,s)

Summary

- introduced eXtensible Dependency Grammar DG(X)
- DG(X): graph description language
- TDG = DG(d,t)
- compositional semantics composition: DG(d,t,s)
- target semantics: CLLS

Outlook

- apply to more languages (already applied to: English, Dutch, German, Czech)

Outlook

- apply to more languages (already applied to: English, Dutch, German, Czech)
- automatic grammar/lexicon generation

Outlook

- apply to more languages (already applied to: English, Dutch, German, Czech)
- automatic grammar/lexicon generation
- interface to preferences (e.g. free modifiers or scope)