XDG - eXtensible Dependency Grammar

Ralph Debusmann

rade@ps.uni-sb.de

Programming Systems Lab

Universität des Saarlandes

Overview

Overview

1. Introduction
2. Introducing XDG
3. First instance: TDG
4. Second instance: TDGS
5. Syntax-semantics interface to CLLS
6. Conclusion

Introduction

- idea: parse natural language utterances and construct its corresponding semantic representation
- i.e. our goal is a function f from a string of words W^* to a set of semantic representations S:

$$f \quad : \quad W^* \longrightarrow 2^S$$

- we specify f using a *grammar formalism*

Existing grammar formalisms

- most popular: formalisms based on context-free grammar, e.g:
  - LFG (Bresnan/Kaplan 82)
  - GB (Chomsky 86)
  - TAG (Joshi 87)
  - HPSG (Pollard/Sag 94)
- less popular: formalisms based on dependency grammar, e.g.:
  - FGD (Sgall 86)*
  - MTT (Melcuk 88)*
  - WG (Hudson 90)*
  - TDG (Duchier/Debusmann 01)*
- why? no syntax-semantics interface

Towards a syntax-semantics interface for TDG

- two steps towards a syntax-semantics interface for TDG:
  1. generalize TDG to a meta grammar formalism: XDG (eXtensible Dependency Grammar)
  2. instantiate XDG to obtain a grammar formalism with a syntax-semantics interface (TDGS)

Overview

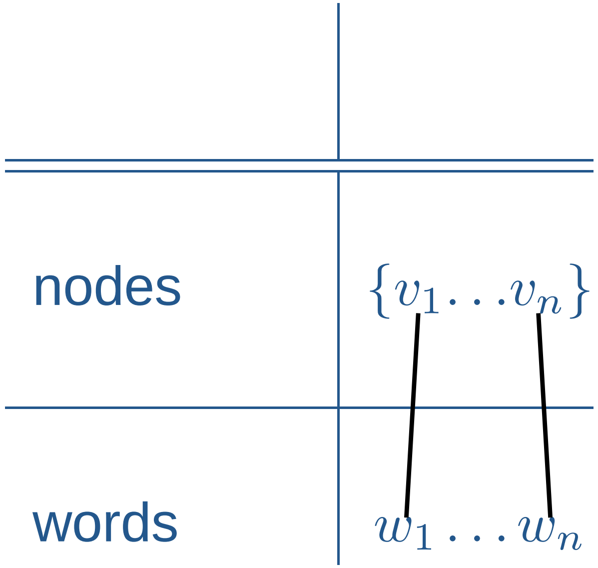
1. Introduction
2. Introducing XDG
3. First instance: TDG
4. Second instance: TDGS
5. Syntax-semantics interface to CLLS
6. Conclusion

XDG architecture

words	$w_1 \ldots w_n$

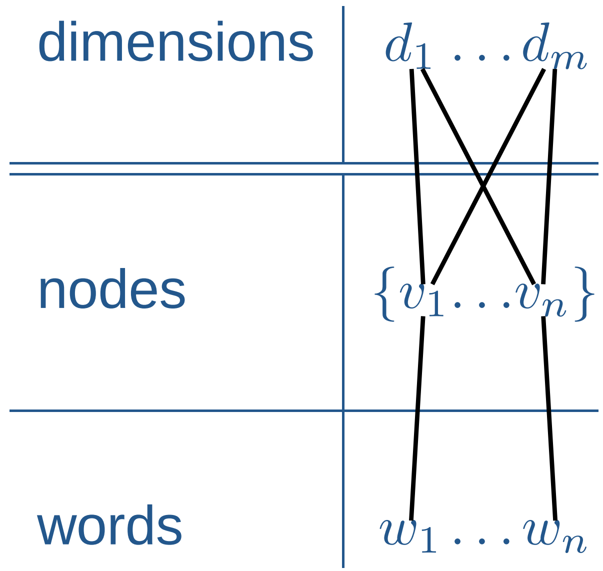
- the input string of an XDG analysis $w_1 \ldots w_n$ consists of words w from the set of words W

XDG architecture



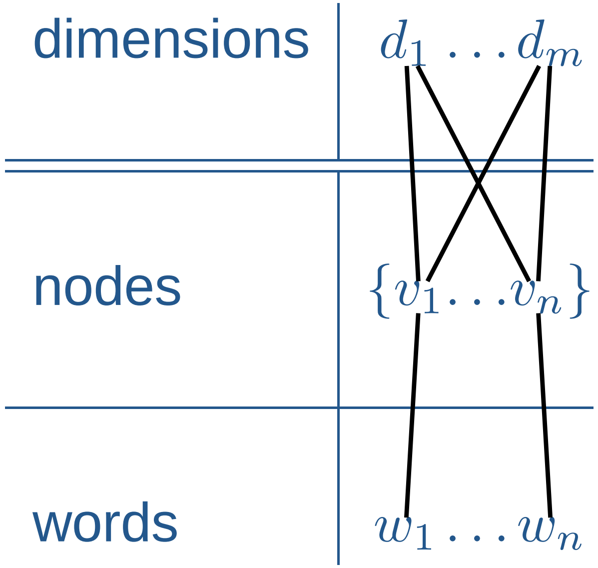
- the words $w_1, \ldots, w_n$ correspond one-to-one to nodes $v_1 \ldots v_n$ in node set V

XDG architecture



- these nodes $v_1 \ldots v_n$ are shared across the m dimensions $D = \{d_1, \ldots, d_m\}$

XDG architecture



- these nodes $v_1 \ldots v_n$ are shared across the m dimensions $D = \{d_1, \ldots, d_m\}$
- a dimension $d \in D$ corresponds to a directed labeled graph (V, E_d), where $E_d = V \times V \times \mathcal{L}_d$. $\mathcal{L}_d$ is the set of edge labels on dimension d.

Principles

- each dimension d is subject to a set P_d of *principles*

Principles

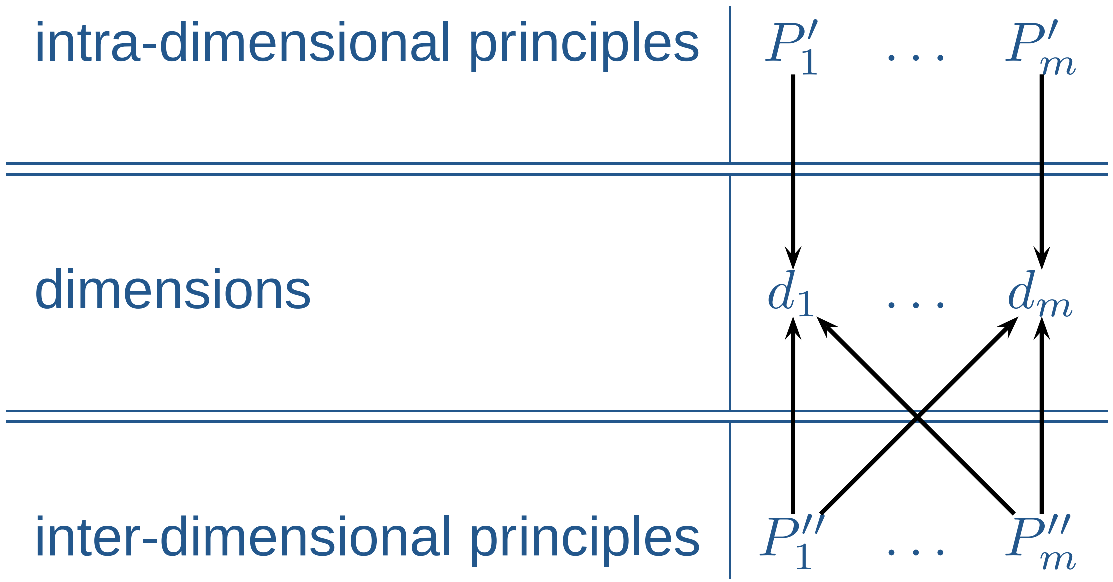
- each dimension d is subject to a set P_d of *principles*
- the principles are drawn from a shared *principles pool* P

Principles

- each dimension d is subject to a set P_d of *principles*
- the principles are drawn from a shared *principles pool* P
- we partition P into the set of intra-dimensional principles P' (apply only within one dimension) and the set of inter-dimensional principles P'' (apply across several dimensions):

Principles

- each dimension d is subject to a set P_d of *principles*
- the principles are drawn from a shared *principles pool* P
- we partition P into the set of intra-dimensional principles P' (apply only within one dimension) and the set of inter-dimensional principles P'' (apply across several dimensions):



Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\mathrm{dag}(d : D)$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\text{dag}(d : D)$$
$$\text{tree}(d : D)$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\mathrm{dag}(d : D)$$
$$\mathrm{tree}(d : D)$$
$$\mathrm{out}(d : D, \mathsf{out_d} : V \to 2^{\mathcal{L}'_d})$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\mathrm{dag}(d : D)$$
$$\mathrm{tree}(d : D)$$
$$\mathrm{out}(d : D, \mathsf{out_d} : V \to 2^{\mathcal{L}'_d})$$
$$\mathrm{in}(d : D, \mathsf{in_d} : V \to 2^{\mathcal{L}'_d})$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\mathrm{dag}(d : D)$$
$$\mathrm{tree}(d : D)$$
$$\mathrm{out}(d : D, \mathsf{out_d} : V \rightarrow 2^{\mathcal{L}'_d})$$
$$\mathrm{in}(d : D, \mathsf{in_d} : V \rightarrow 2^{\mathcal{L}'_d})$$
$$\mathrm{order}(d : D, N_d, \mathsf{on_d} : V \rightarrow N_d, \prec_d)$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$\text{dag}(d : D)$$
$$\text{tree}(d : D)$$
$$\text{out}(d : D, \mathsf{out_d} : V \to 2^{\mathcal{L}'_d})$$
$$\text{in}(d : D, \mathsf{in_d} : V \to 2^{\mathcal{L}'_d})$$
$$\text{order}(d : D, N_d, \mathsf{on_d} : V \to N_d, \prec_d)$$
$$\text{climbing}(d_1 : D, d_2 : D)$$

Principles and parameters

- here is the principles pool for this talk. The principles are parametrized e.g. by the dimension on which they apply and functions with domain V, called *features*:

$$
\begin{array}{l}
\mathrm{dag}(d : D) \\
\mathrm{tree}(d : D) \\
\mathrm{out}(d : D, \mathsf{out_d} : V \rightarrow 2^{\mathcal{L}'_d}) \\
\mathrm{in}(d : D, \mathsf{in_d} : V \rightarrow 2^{\mathcal{L}'_d}) \\
\mathrm{order}(d : D, N_d, \mathsf{on_d} : V \rightarrow N_d, \prec_d) \\
\mathrm{climbing}(d_1 : D, d_2 : D) \\
\mathrm{linking}(d_1 : D, d_2 : D, \mathsf{link_{d_1}} : V \rightarrow (\mathcal{L}_{d_1} \rightarrow 2^{\mathcal{L}_{d_2}}))
\end{array}
$$

Dag principle

$\mathrm{dag}(d : D)$: Each analysis on dimension d is a directed acyclic graph.

Tree principle

$\mathrm{tree}(d : D)$: Each analysis on dimension d is a tree.

Out principle

$\mathrm{out}(d : D, \mathtt{out_d} : V \to 2^{\mathcal{L}'_d})$: The outgoing edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding $\mathtt{out}_d$ feature.

Out principle

$\mathrm{out}(d : D, \mathsf{out_d} : V \rightarrow 2^{\mathcal{L}'_d})$: The outgoing edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding $\mathtt{out}_d$ feature.

- here, $\mathcal{L}'_d$ is the set of label patterns ℓ', defined by the following abstract syntax:

$$\ell' \quad ::= \quad \ell \mid \ell? \mid \ell* \qquad\qquad \ell \in \mathcal{L}_d$$

Out principle

$\mathrm{out}(d : D, \mathtt{out_d} : V \to 2^{\mathcal{L}'_d})$: The outgoing edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding $\mathtt{out}_d$ feature.

- here, $\mathcal{L}'_d$ is the set of label patterns ℓ', defined by the following abstract syntax:

$$\ell' \quad ::= \quad \ell \mid \ell? \mid \ell* \qquad\qquad \ell \in \mathcal{L}_d$$

- ℓ: precisely one outgoing edge, $\ell?$: zero or one, $\ell*$: zero or more

In principle

$\mathrm{in}(d : D, \mathsf{in_d} : V \rightarrow 2^{\mathcal{L}'_d})$: The incoming edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding in_d feature.

In principle

$\mathrm{in}(d : D, \mathsf{in_d} : V \rightarrow 2^{\mathcal{L}'_d})$: The incoming edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding in_d feature.

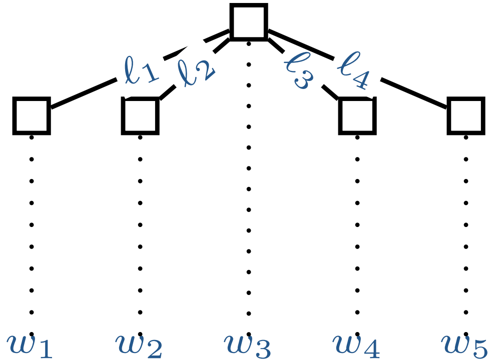
- symmetrical to the out principle

Order principle

$\mathrm{order}(d : D, N_d, \mathsf{on_d} : V \to N_d, \prec_d)$: The daughters of a node on dimension d must be ordered according to their edge label and the total order stipulated in $\prec_d$. The node itself is assigned a node label by the $\mathsf{on_d}$ feature, by which it is positioned with respect to its daughters.

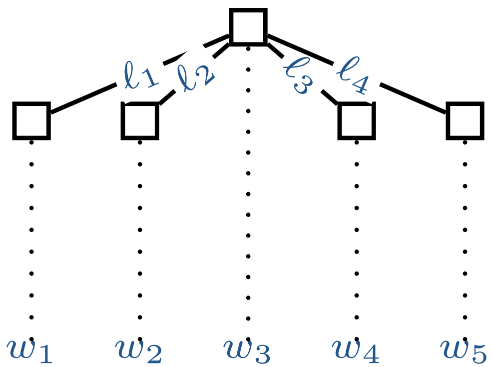
Order principle: intuition

- consider:



Order principle: intuition

- consider:



- assuming $\prec_d = \ell_1 \prec \ell_2 \prec \ell_3 \prec \ell_4$, the analysis is well-formed, since:

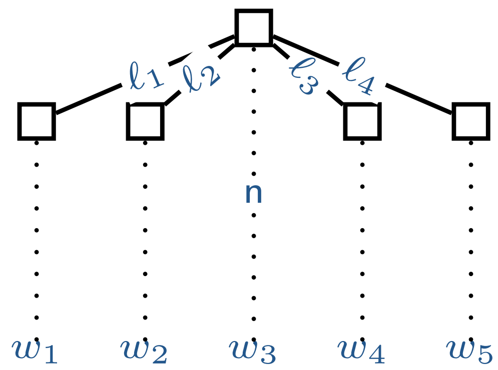
$$\begin{array}{ccccccc} \ell_1 & \prec & \ell_2 & \prec & \ell_3 & \prec & \ell_4 \\ w_1 & < & w_2 & < & w_4 & < & w_5 \end{array}$$

Order principle: intuition

- we still have to position w_3 with respect to its daughters

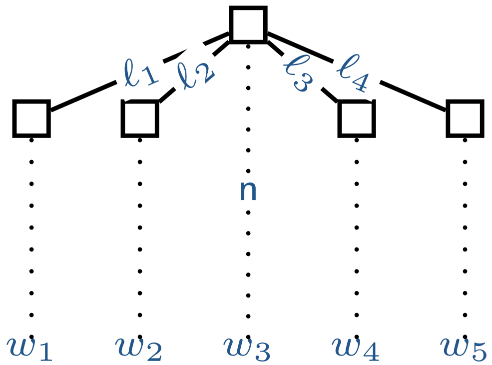
Order principle: intuition

- we still have to position w_3 with respect to its daughters
- assume $\mathsf{on}_d(v_3) = \mathsf{n}$:



Order principle: intuition

- we still have to position w_3 with respect to its daughters
- assume $\mathsf{on}_d(v_3) = \mathsf{n}$:



- assuming $\prec_d = \ell_1 \prec \ell_2 \prec \mathsf{n} \prec \ell_3 \prec \ell_4$, the analysis is well-formed, since:

$$\begin{array}{ccccccccc} \ell_1 & \prec & \ell_2 & \prec & \mathsf{n} & \prec & \ell_3 & \prec & \ell_4 \\ w_1 & < & w_2 & < & w_3 & < & w_4 & < & w_5 \end{array}$$

Climbing principle

$\text{climbing}(d_1 : D, d_2 : D)$: A node on dimension d_1 may climb up and land higher up on dimension d_2.

Linking principle

$\mathrm{linking}(d_1 : D, d_2 : D, \mathsf{link}_{\mathsf{d}_1} : V \to (\mathcal{L}_{d_1} \to 2^{\mathcal{L}_{d_2}}))$: An edge $v_1 - \ell_1 \to_{d_1} v_2$ on dimension d_1 is licensed only if v_2 has incoming edge label $\ell_2 \in \mathsf{link}_{d_1}(v_1)(\ell_1)$ on dimension d_2.

Lexical features

- features $\mathsf{f} : V \to X$ are defined by means of the *lexicon*

Lexical features

- features $\mathsf{f} : V \to X$ are defined by means of the *lexicon*
- the lexicon is a function $\mathsf{l} : W \to 2^E$ mapping words to sets of lexical entries

Lexical features

- features $\mathsf{f} : V \to X$ are defined by means of the *lexicon*
- the lexicon is a function $\mathsf{l} : W \to 2^E$ mapping words to sets of lexical entries
- in an analysis, only one lexical entry is assigned to each node by selection function $\mathsf{s} : 2^E \to E$

Lexical features

- features $\mathsf{f} : V \rightarrow X$ are defined by means of the *lexicon*
- the lexicon is a function $\mathsf{l} : W \rightarrow 2^E$ mapping words to sets of lexical entries
- in an analysis, only one lexical entry is assigned to each node by selection function $\mathsf{s} : 2^E \rightarrow E$
- lexical entries are records describing functions $\mathsf{f}' : E \rightarrow X$

Lexical features

- features $\mathsf{f} : V \rightarrow X$ are defined by means of the *lexicon*
- the lexicon is a function $\mathsf{l} : W \rightarrow 2^E$ mapping words to sets of lexical entries
- in an analysis, only one lexical entry is assigned to each node by selection function $\mathsf{s} : 2^E \rightarrow E$
- lexical entries are records describing functions $\mathsf{f}' : E \rightarrow X$
- f can now be defined as follows:

$$\mathsf{f}(v) \quad : \quad \mathsf{f}'(\mathsf{s}(\mathsf{l}(w)))$$

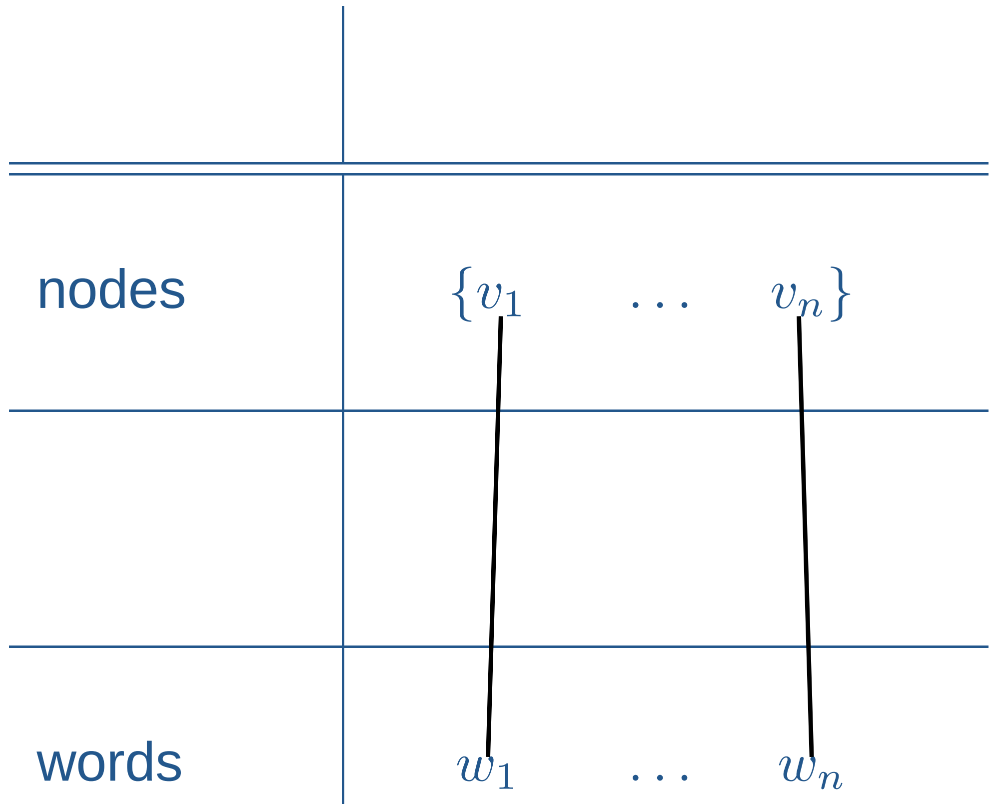
where word w corresponds to node v.

XDG architecture revisited

words	w_1 $\dots$ w_n

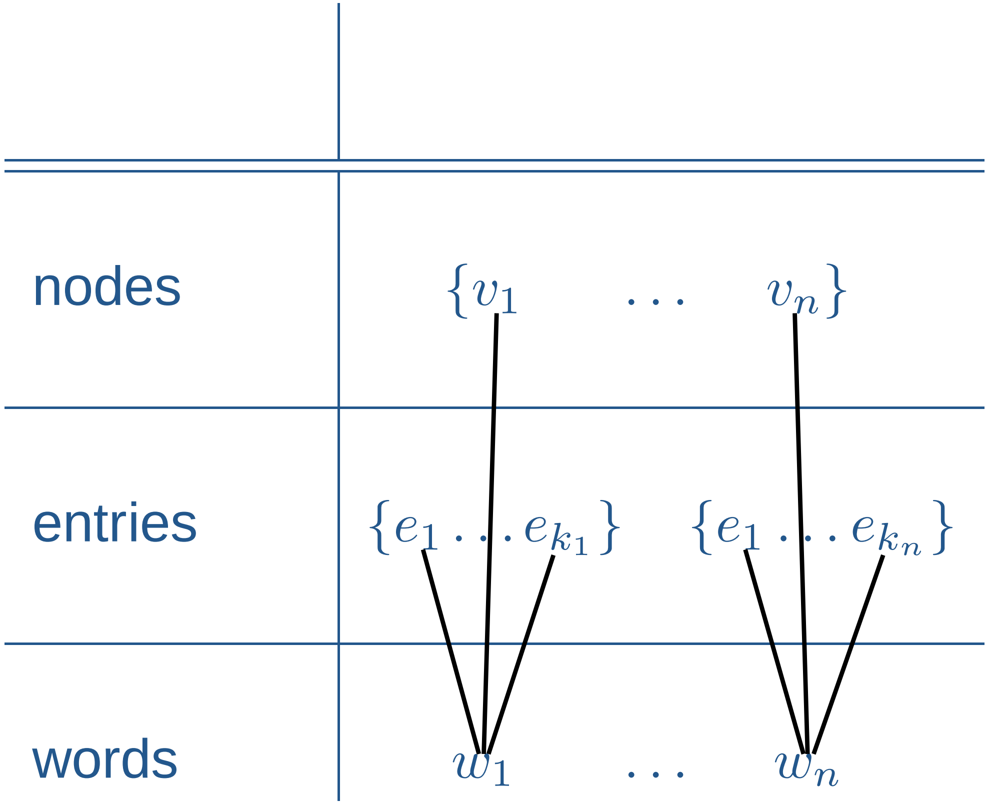
- the input string $w_1 \dots w_n$ consists of words $w \in W$

XDG architecture revisited



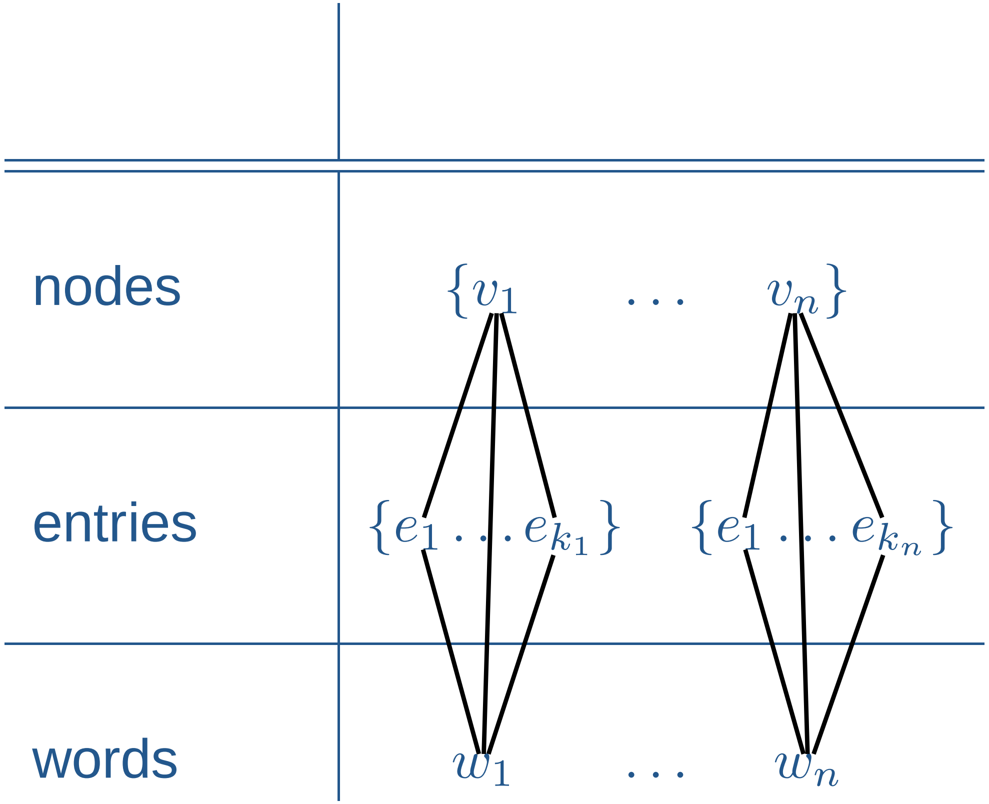
- these words correspond one-to-one to nodes $v_1 \ldots v_n$ in node set V

XDG architecture revisited



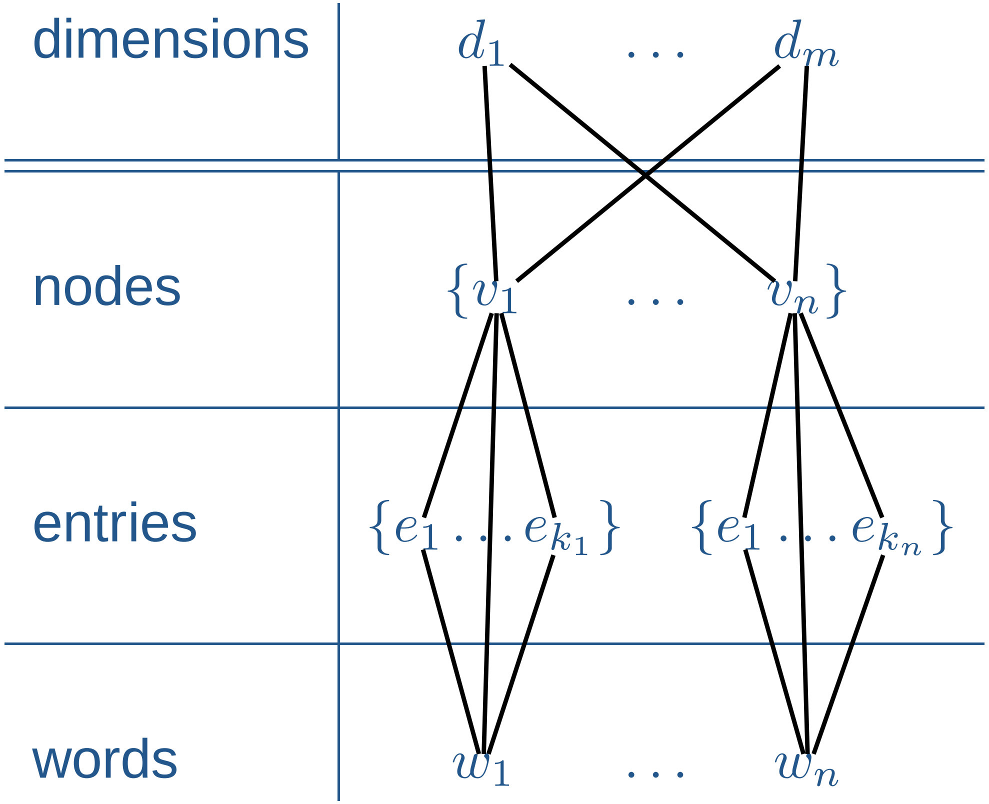
- each word w_i in the input string is assigned a set of lexical entries $\{e_1, \ldots, e_{k_i}\}$

XDG architecture revisited



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

XDG architecture revisited



- the node set V is shared across the m dimensions $D = \{d_1, \dots, d_m\}$

Overview

1. Introduction
2. Introducing XDG
3. First instance: TDG
4. Second instance: TDGS
5. Syntax-semantics interface to CLLS
6. Conclusion

TDG grammar

- now we can instantiate XDG to obtain a TDG grammar (Duchier/Debusmann 01):

TDG grammar

- now we can instantiate XDG to obtain a TDG grammar (Duchier/Debusmann 01):

$$
\begin{aligned}
D &= \{\mathsf{ID}, \mathsf{LP}\} \\
\\
P_{\mathsf{ID}} &= \{\mathrm{tree}(\mathsf{ID}), \mathrm{out}(\mathsf{ID}, \mathsf{out}_{\mathsf{ID}}), \mathrm{in}(\mathsf{ID}, \mathsf{in}_{\mathsf{ID}})\} \\
\mathcal{L}_{\mathsf{ID}} &= \{\mathsf{subj}, \mathsf{obj}, \mathsf{vinf}\} \\
\\
P_{\mathsf{LP}} &= \{\mathrm{tree}(\mathsf{LP}), \mathrm{out}(\mathsf{LP}, \mathsf{out}_{\mathsf{LP}}), \mathrm{in}(\mathsf{LP}, \mathsf{in}_{\mathsf{LP}}), \\
&\quad \mathrm{order}(\mathsf{LP}, N_{\mathsf{LP}}, \mathsf{on}_{\mathsf{LP}}, \prec_{\mathsf{LP}}), \mathrm{climbing}(\mathsf{ID}, \mathsf{LP})\} \\
\mathcal{L}_{\mathsf{LP}} &= \{\mathsf{mf}, \mathsf{vcf}\} \\
N_{\mathsf{LP}} &= \{\mathsf{n}, \mathsf{v}\} \\
\prec_{\mathsf{LP}} &= \mathsf{n} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v}
\end{aligned}
$$

TDG

- or, more intuitively:

TDG

- or, more intuitively:

intra-dimensional principles		
dimensions	ID	LP
inter-dimensional principles		

TDG

- or, more intuitively:

intra-dimensional principles	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}\}$	
dimensions	ID	LP
inter-dimensional principles		

TDG

- or, more intuitively:

intra-dimensional principles	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}\}$	
dimensions	ID	LP
inter-dimensional principles	$\emptyset$	

TDG

- or, more intuitively:

intra-dimensional principles	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}\}$ ↓	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}, \mathrm{order}\}$ ↓
dimensions	ID	LP
inter-dimensional principles	$\emptyset$	

TDG

- or, more intuitively:

intra-dimensional principles	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}\}$	$\{\mathrm{tree}, \mathrm{in}, \mathrm{out}, \mathrm{order}\}$
dimensions	ID	LP
inter-dimensional principles	$\emptyset$	$\{\mathrm{climbing}\}$

TDG analysis

- consists of two dimensions: ID and LP

TDG analysis

- consists of two dimensions: ID and LP
- the models on both dimensions must be trees (ID tree, LP tree)

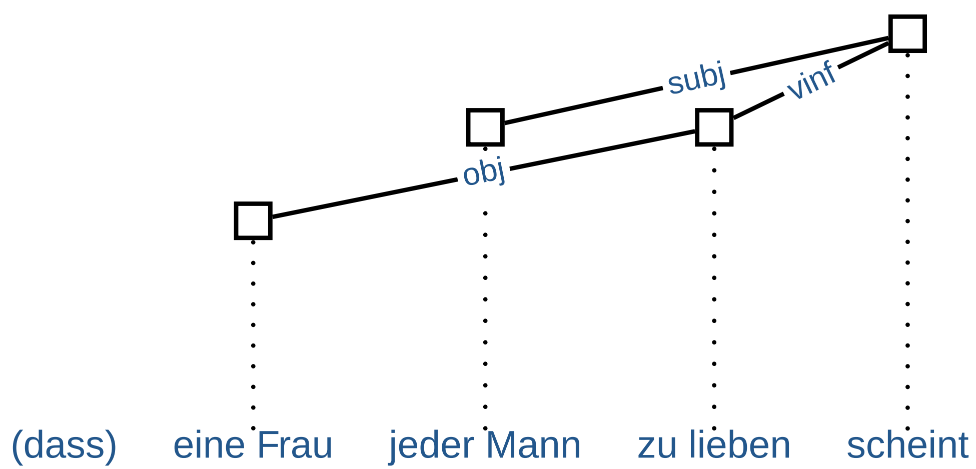
TDG analysis

- consists of two dimensions: ID and LP
- the models on both dimensions must be trees (ID tree, LP tree)
- the ID tree is unordered, the LP tree is ordered by the order principle

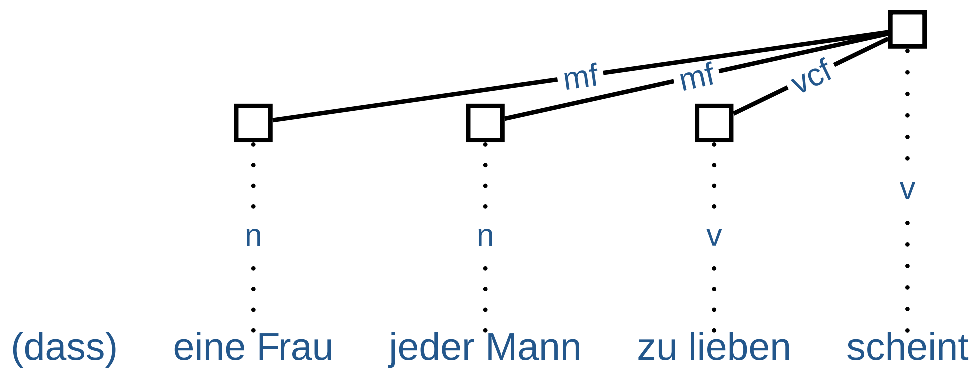
TDG analysis

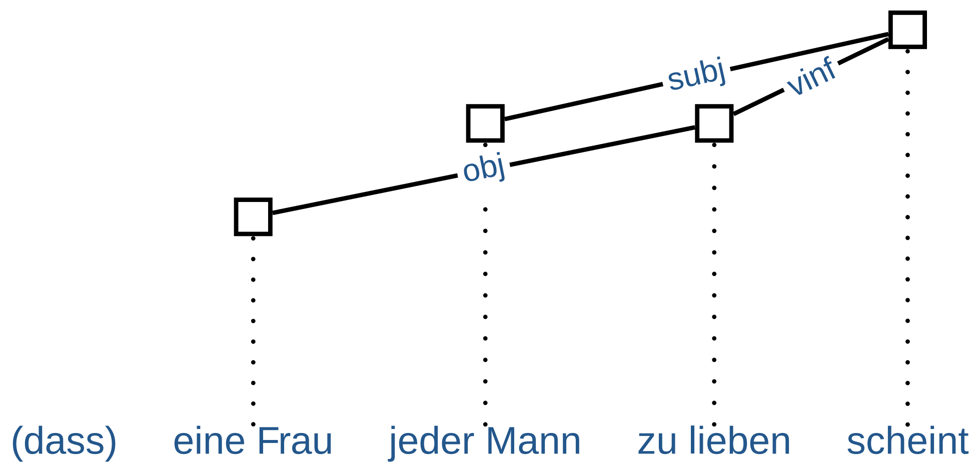
- consists of two dimensions: ID and LP
- the models on both dimensions must be trees (ID tree, LP tree)
- the ID tree is unordered, the LP tree is ordered by the order principle
- the LP tree is a flattening of the ID tree by virtue of the climbing principle

Example

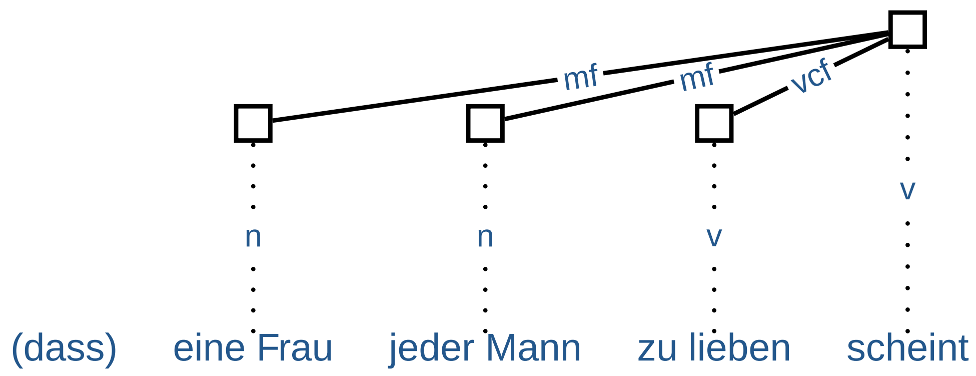


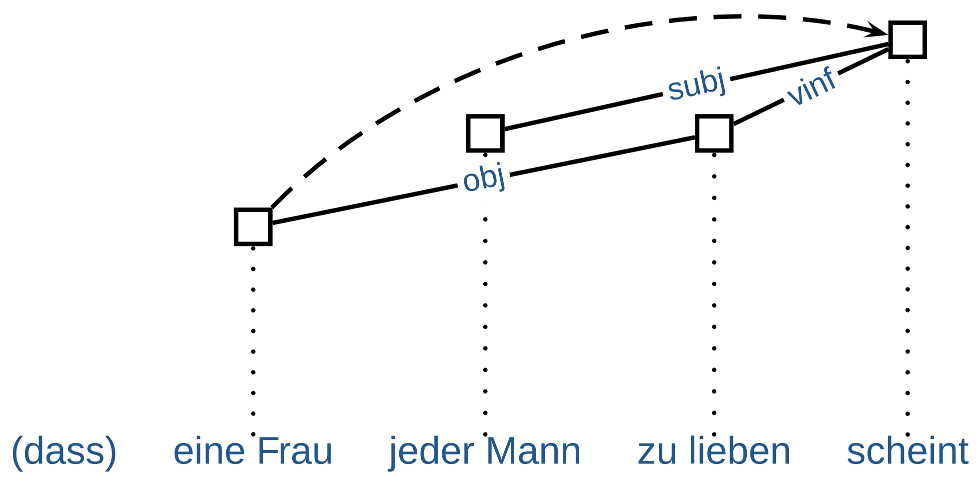
Example


mf
mf
vcf
n
n
v
v
(dass) eine Frau jeder Mann zu lieben scheint


subj
vinf
obj
(dass) eine Frau jeder Mann zu lieben scheint

Example


mf
mf
vcf
n
n
v
v
(dass) eine Frau jeder Mann zu lieben scheint


subj
vinf
obj
(dass) eine Frau jeder Mann zu lieben scheint

Lexical entry signature

- includes all features required by the TDG principles:

Lexical entry signature

- includes all features required by the TDG principles:

$$\begin{bmatrix} \text{ID} & : & \begin{bmatrix} \text{in} & : & 2^{\mathcal{L}'_{\text{ID}}} \\ \text{out} & : & 2^{\mathcal{L}'_{\text{ID}}} \end{bmatrix} \\ \text{LP} & : & \begin{bmatrix} \text{in} & : & 2^{\mathcal{L}'_{\text{LP}}} \\ \text{on} & : & N_{\text{LP}} \\ \text{out} & : & 2^{\mathcal{L}'_{\text{LP}}} \end{bmatrix} \end{bmatrix}$$

Example lexicon: nouns

$$jeder\ Mann \mapsto \begin{bmatrix} \mathsf{ID} & : & \begin{bmatrix} \mathsf{in} & : & \{\mathsf{subj}?\} \\ \mathsf{out} & : & \emptyset \end{bmatrix} \\ \mathsf{LP} & : & \begin{bmatrix} \mathsf{in} & : & \{\mathsf{mf}?\} \\ \mathsf{on} & : & \mathsf{n} \\ \mathsf{out} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

Example lexicon: nouns

$$jeder\ Mann \mapsto \begin{bmatrix} \textsf{ID} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{subj?}\} \\ \textsf{out} & : & \emptyset \end{bmatrix} \\ \textsf{LP} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{mf?}\} \\ \textsf{on} & : & \textsf{n} \\ \textsf{out} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

$$eine\ Frau \mapsto \begin{bmatrix} \textsf{ID} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{obj?}\} \\ \textsf{out} & : & \emptyset \end{bmatrix} \\ \textsf{LP} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{mf?}\} \\ \textsf{on} & : & \textsf{n} \\ \textsf{out} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

Example lexicon: verbs

$$
\textit{zu lieben} \mapsto \begin{bmatrix} \textsf{ID} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{vinf}?\} \\ \textsf{out} & : & \{\textsf{obj}\} \end{bmatrix} \\ \textsf{LP} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{vcf}?\} \\ \textsf{on} & : & \textsf{v} \\ \textsf{out} & : & \emptyset \end{bmatrix} \end{bmatrix}
$$

Example lexicon: verbs

$$zu\ lieben \mapsto \begin{bmatrix} \mathsf{ID} & : & \begin{bmatrix} \mathsf{in} & : & \{\mathsf{vinf?}\} \\ \mathsf{out} & : & \{\mathsf{obj}\} \end{bmatrix} \\ \mathsf{LP} & : & \begin{bmatrix} \mathsf{in} & : & \{\mathsf{vcf?}\} \\ \mathsf{on} & : & \mathsf{v} \\ \mathsf{out} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

$$scheint \mapsto \begin{bmatrix} \mathsf{ID} & : & \begin{bmatrix} \mathsf{in} & : & \emptyset \\ \mathsf{out} & : & \{\mathsf{subj}, \mathsf{vinf}\} \end{bmatrix} \\ \mathsf{LP} & : & \begin{bmatrix} \mathsf{in} & : & \emptyset \\ \mathsf{on} & : & \mathsf{v} \\ \mathsf{out} & : & \{\mathsf{mf}*, \mathsf{vcf?}\} \end{bmatrix} \end{bmatrix}$$

Overview

1. Introduction
2. Introducing XDG
3. First instance: TDG
4. Second instance: TDGS
5. Syntax-semantics interface to CLLS
6. Conclusion

TDGS grammar

- next, we prepare the construction of a semantics. Therefore, we instantiate XDG to obtain TDGS (TDG with Semantics):

TDGS grammar

- next, we prepare the construction of a semantics. Therefore, we instantiate XDG to obtain TDGS (TDG with Semantics):

$$D = \{\mathsf{ID}, \mathsf{LP}, \mathsf{TH}\}$$

$$\begin{aligned} P_{\mathsf{ID}} &= \{\mathrm{tree}(\mathsf{ID}), \mathrm{out}(\mathsf{ID}, \mathsf{out}_{\mathsf{ID}}), \mathrm{in}(\mathsf{ID}, \mathsf{in}_{\mathsf{ID}})\} \\ \mathcal{L}_{\mathsf{ID}} &= \{\mathsf{subj}, \mathsf{obj}, \mathsf{vinf}\} \end{aligned}$$

$$\begin{aligned} P_{\mathsf{LP}} &= \{\mathrm{tree}(\mathsf{LP}), \mathrm{out}(\mathsf{LP}, \mathsf{out}_{\mathsf{LP}}), \mathrm{in}(\mathsf{LP}, \mathsf{in}_{\mathsf{LP}}), \\ &\quad \mathrm{order}(\mathsf{LP}, N_{\mathsf{LP}}, \mathsf{on}_{\mathsf{LP}}, \prec_{\mathsf{LP}}), \mathrm{climbing}(\mathsf{ID}, \mathsf{LP})\} \\ \mathcal{L}_{\mathsf{LP}} &= \{\mathsf{mf}, \mathsf{vcf}\} \\ N_{\mathsf{LP}} &= \{\mathsf{n}, \mathsf{v}\} \\ \prec_{\mathsf{LP}} &= \mathsf{n} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v} \end{aligned}$$

TDGS grammar

- next, we prepare the construction of a semantics. Therefore, we instantiate XDG to obtain TDGS (TDG with Semantics):

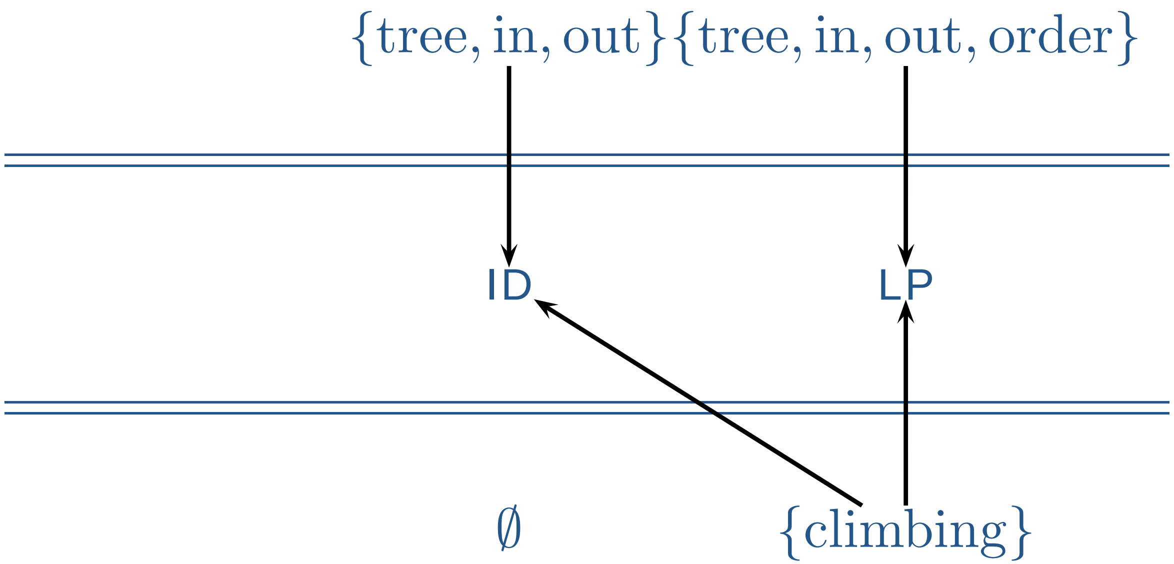
$$
\begin{aligned}
D &= \{\mathsf{ID}, \mathsf{LP}, \mathsf{TH}\} \\
P_{\mathsf{TH}} &= \{\mathrm{dag}(\mathsf{TH}), \mathrm{out}(\mathsf{TH}, \mathsf{out}_{\mathsf{TH}}), \mathrm{in}(\mathsf{TH}, \mathsf{in}_{\mathsf{TH}}), \\
&\quad\ \mathrm{climbing}(\mathsf{TH}, \mathsf{ID}), \mathrm{linking}(\mathsf{TH}, \mathsf{ID}, \mathsf{link}_{\mathsf{TH}}\} \\
\mathcal{L}_{\mathsf{TH}} &= \{\mathsf{act}, \mathsf{pat}, \mathsf{prop}\}
\end{aligned}
$$

TDGS

- or, more intuitively:

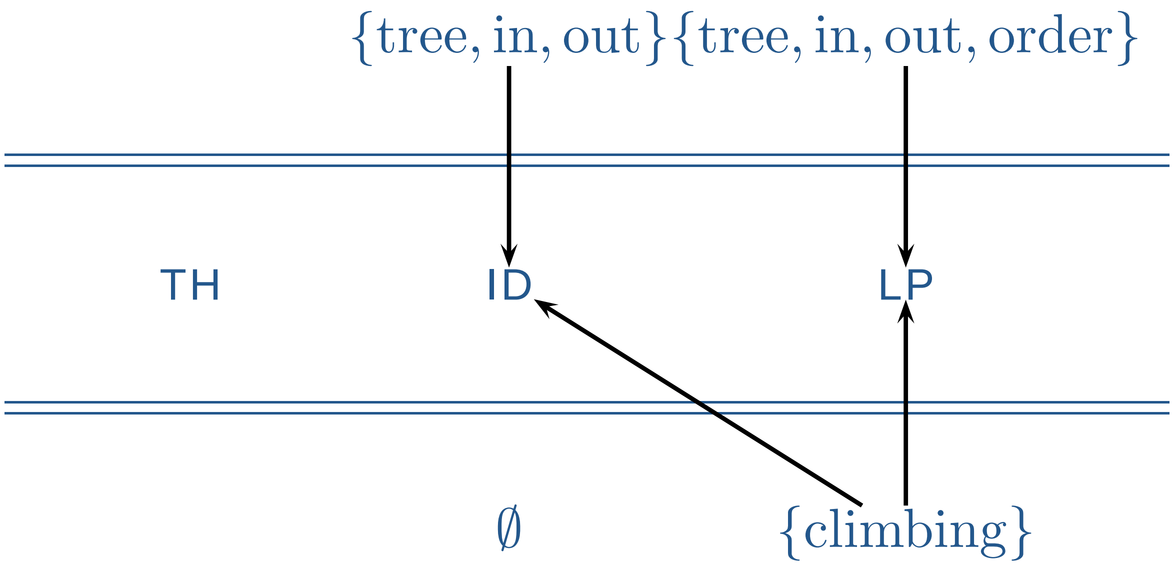
TDGS

- or, more intuitively:



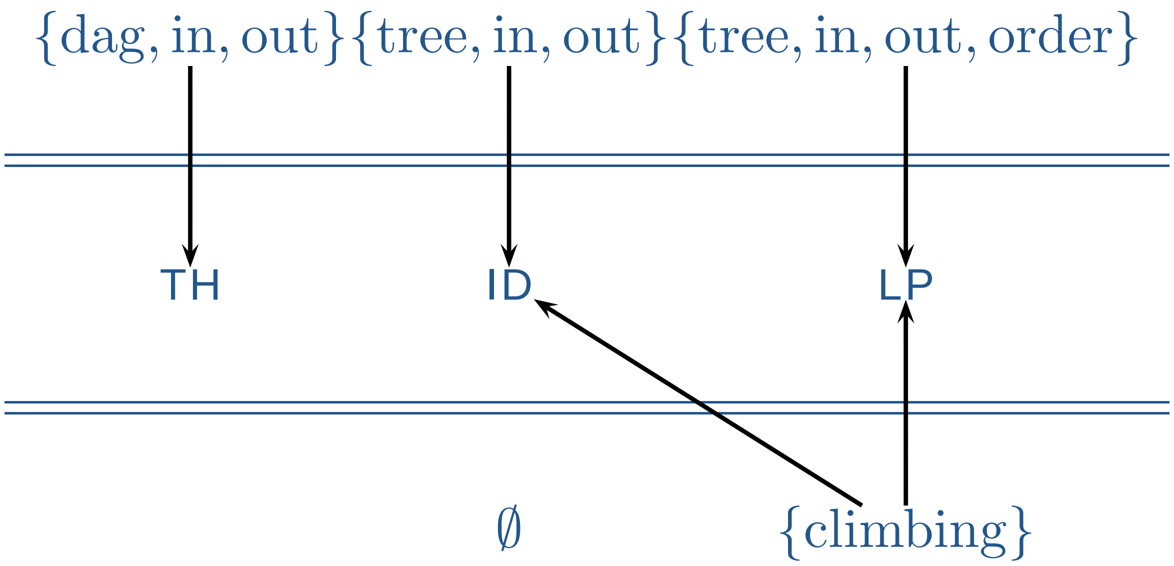
TDGS

- or, more intuitively:



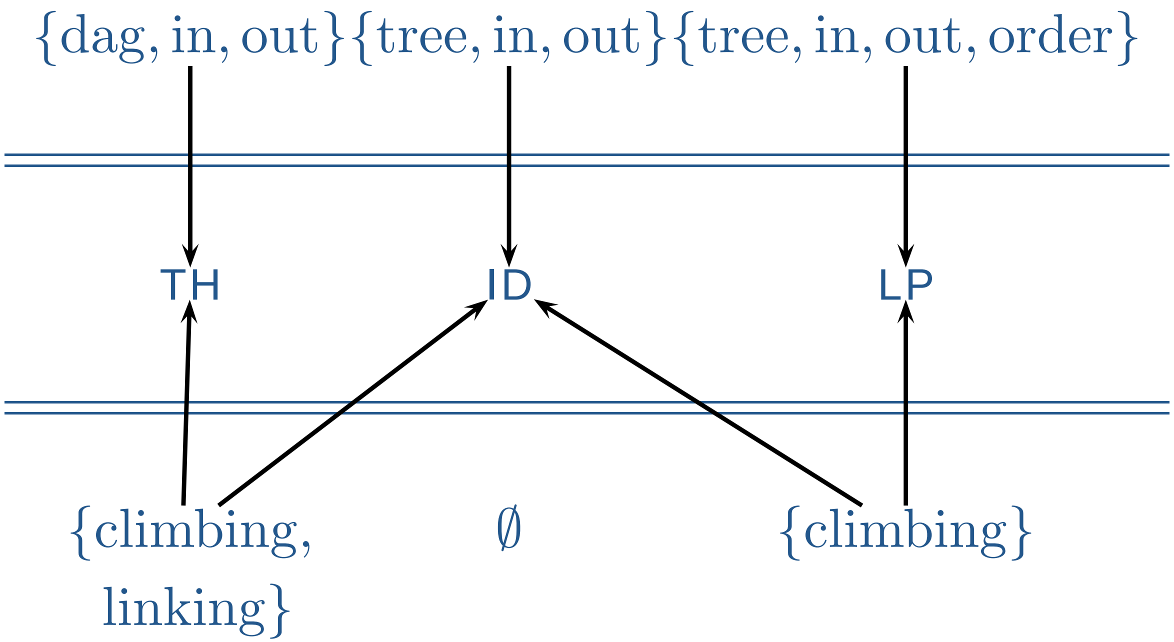
TDGS

- or, more intuitively:



TDGS

- or, more intuitively:



TDGS analysis

- consists of an ID tree, an LP tree and a TH dag

TDGS analysis

- consists of an ID tree, an LP tree and a TH dag
- ID and LP tree as in TDG

TDGS analysis

- consists of an ID tree, an LP tree and a TH dag
- ID and LP tree as in TDG
- the TH dag is a directed acyclic graph reflecting semantic argument structure, rather than syntactic argument structure

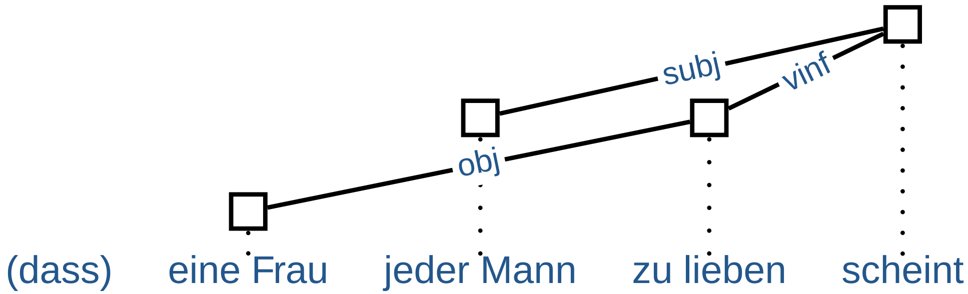
TDGS analysis

- consists of an ID tree, an LP tree and a TH dag
- ID and LP tree as in TDG
- the TH dag is a directed acyclic graph reflecting semantic argument structure, rather than syntactic argument structure
- the LP tree is a flattening of the ID tree by virtue of the climbing principle

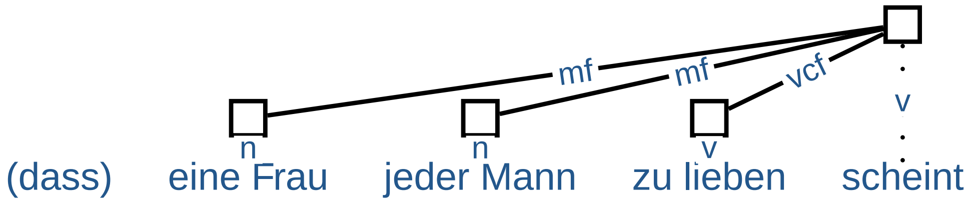
TDGS analysis

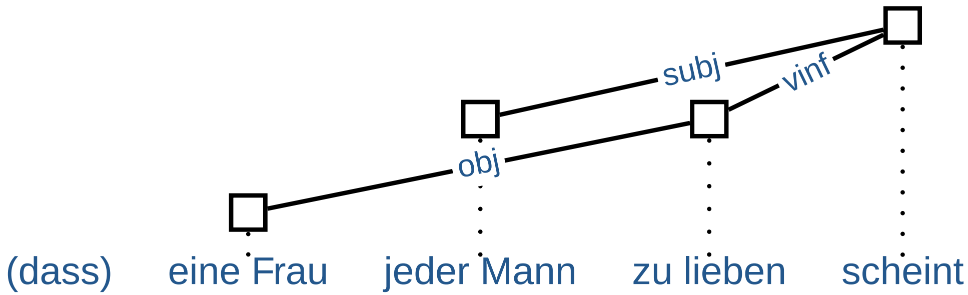
- consists of an ID tree, an LP tree and a TH dag
- ID and LP tree as in TDG
- the TH dag is a directed acyclic graph reflecting semantic argument structure, rather than syntactic argument structure
- the LP tree is a flattening of the ID tree by virtue of the climbing principle
- the ID tree is a flattening of the TH dag, again by virtue of the climbing principle

Example

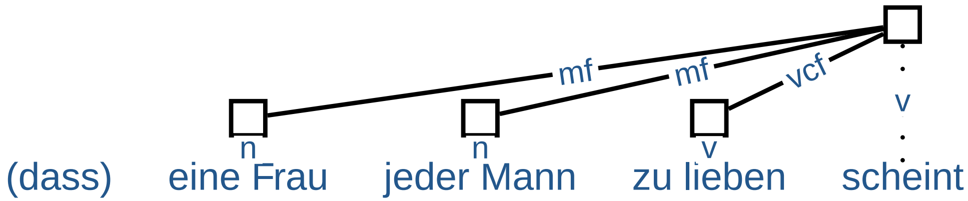


Example

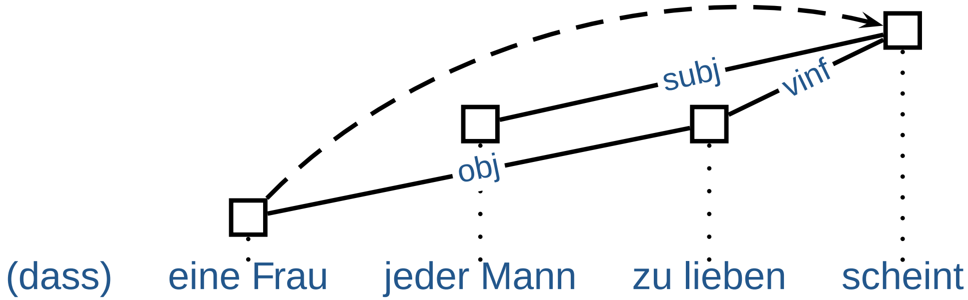

mf
mf
vcf
v
n
n
v
(dass)
eine Frau
jeder Mann
zu lieben
scheint


subj
vinf
obj
(dass)
eine Frau
jeder Mann
zu lieben
scheint

Example

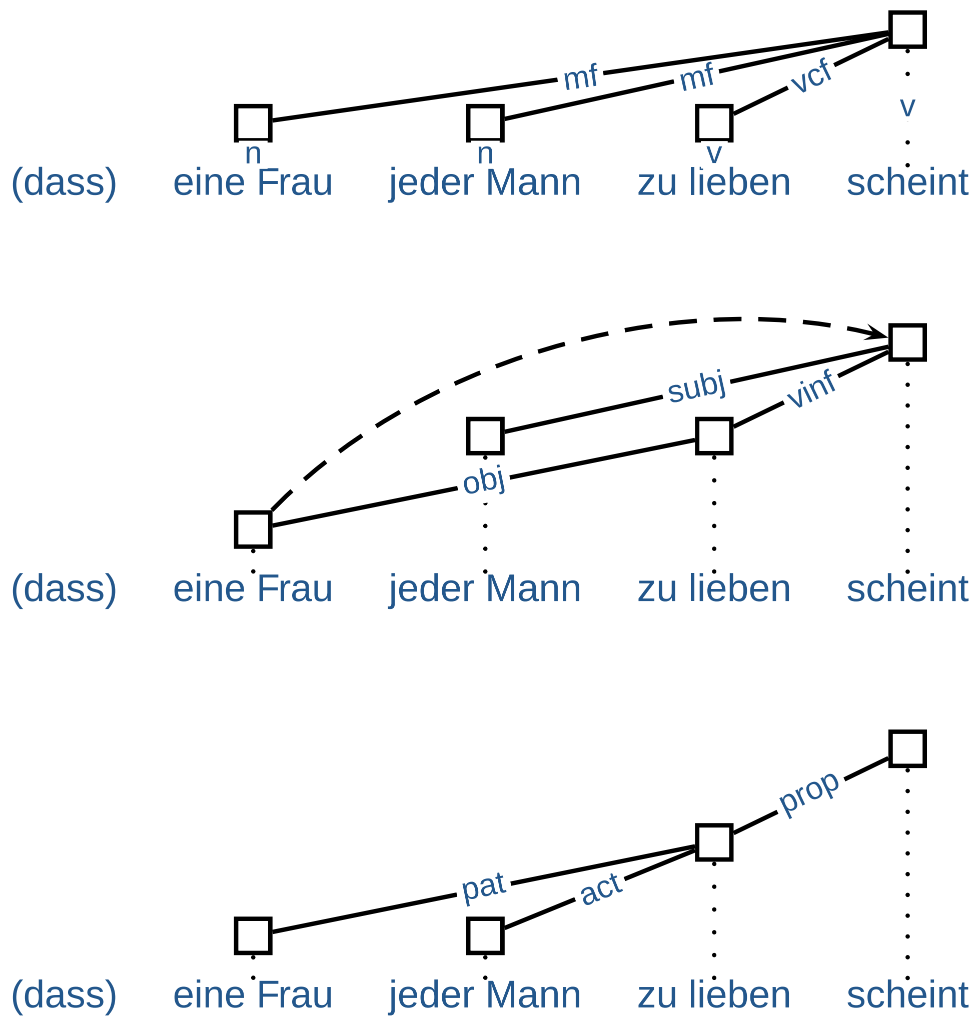


mf
mf
vcf
v
n
n
v
(dass) eine Frau jeder Mann zu lieben scheint

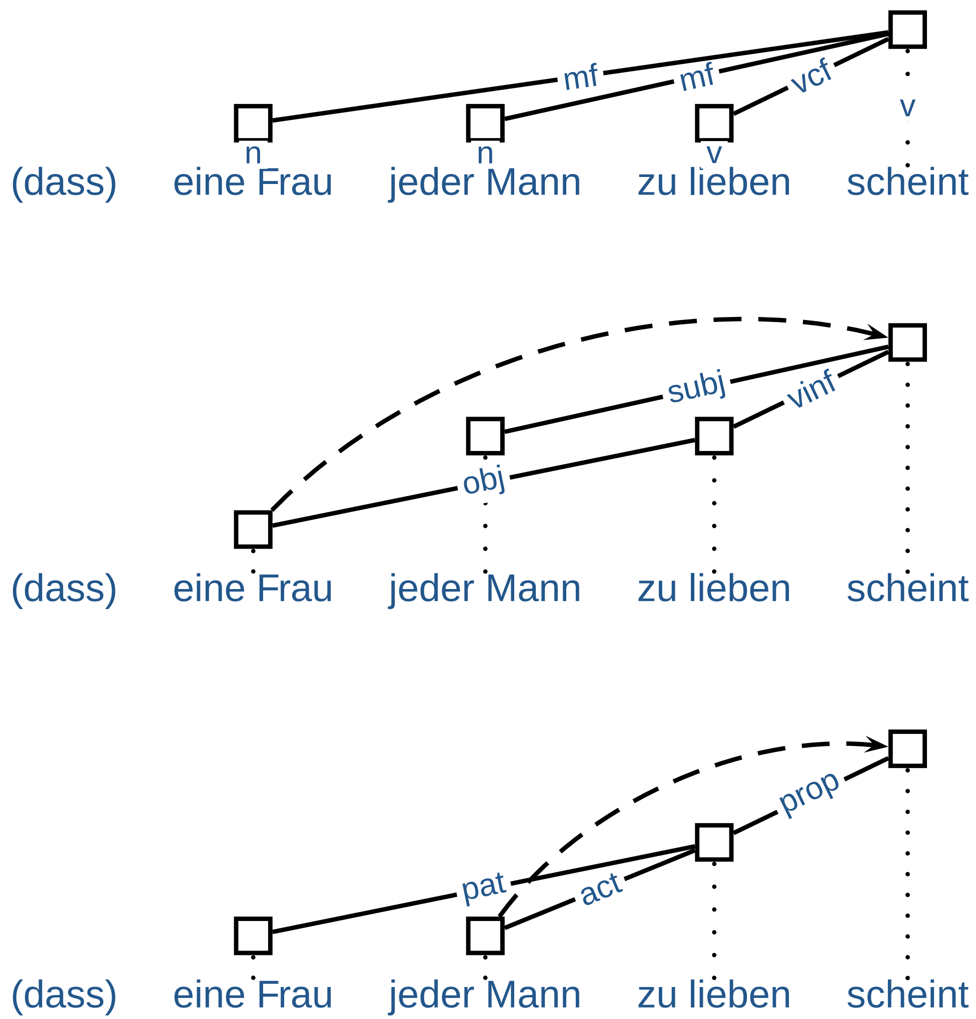


subj
vinf
obj
(dass) eine Frau jeder Mann zu lieben scheint

Example


mf
mf
vcf
n
n
v
v
(dass)
eine Frau
jeder Mann
zu lieben
scheint
subj
vinf
obj
(dass)
eine Frau
jeder Mann
zu lieben
scheint
prop
pat
act
(dass)
eine Frau
jeder Mann
zu lieben
scheint

Example



mf
mf
vcf
v
n
n
v
(dass) eine Frau jeder Mann zu lieben scheint
subj
vinf
obj
(dass) eine Frau jeder Mann zu lieben scheint
prop
pat
act
(dass) eine Frau jeder Mann zu lieben scheint

Lexical entry signature

$$\begin{bmatrix} \mathsf{ID} & : & \begin{bmatrix} \mathsf{in} & : & 2^{\mathcal{L}'_{\mathsf{ID}}} \\ \mathsf{out} & : & 2^{\mathcal{L}'_{\mathsf{ID}}} \end{bmatrix} \\ \mathsf{LP} & : & \begin{bmatrix} \mathsf{in} & : & 2^{\mathcal{L}'_{\mathsf{LP}}} \\ \mathsf{on} & : & N_{\mathsf{LP}} \\ \mathsf{out} & : & 2^{\mathcal{L}'_{\mathsf{LP}}} \end{bmatrix} \\ \mathsf{TH} & : & \begin{bmatrix} \mathsf{in} & : & 2^{\mathcal{L}'_{\mathsf{TH}}} \\ \mathsf{out} & : & 2^{\mathcal{L}'_{\mathsf{TH}}} \\ \mathsf{link} & : & \mathcal{L}_{\mathsf{TH}} \longrightarrow 2^{\mathcal{L}_{\mathsf{ID}}} \end{bmatrix} \end{bmatrix}$$

Example lexicon: nouns

$$jeder\ Mann \mapsto \begin{bmatrix} \mathsf{TH} & : & \begin{bmatrix} \mathsf{in} & : & \{\mathsf{act}?\} \\ \mathsf{out} & : & \emptyset \\ \mathsf{link} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

Example lexicon: nouns

$$jeder\ Mann \mapsto \begin{bmatrix} \textsf{TH} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{act?}\} \\ \textsf{out} & : & \emptyset \\ \textsf{link} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

$$eine\ Frau \mapsto \begin{bmatrix} \textsf{TH} & : & \begin{bmatrix} \textsf{in} & : & \{\textsf{pat?}\} \\ \textsf{out} & : & \emptyset \\ \textsf{link} & : & \emptyset \end{bmatrix} \end{bmatrix}$$

Example lexicon: verbs

$$zu\ lieben \mapsto \left[\mathsf{TH} : \left[\begin{array}{rcl} \mathsf{in} & : & \{\mathsf{prop}?\} \\ \mathsf{out} & : & \{\mathsf{act}, \mathsf{pat}\} \\ \mathsf{link} & : & \{\mathsf{act} \mapsto \{\mathsf{subj}\}, \mathsf{pat} \mapsto \{\mathsf{obj}\}\} \end{array} \right] \right]$$

Example lexicon: verbs

$$
\textit{zu lieben} \mapsto
\left[\textsf{TH} : \left[\begin{array}{rcl} \textsf{in} & : & \{\textsf{prop}?\} \\ \textsf{out} & : & \{\textsf{act}, \textsf{pat}\} \\ \textsf{link} & : & \{\textsf{act} \mapsto \{\textsf{subj}\}, \textsf{pat} \mapsto \{\textsf{obj}\}\} \end{array} \right] \right]
$$

$$
\textit{scheint} \mapsto
\left[\textsf{TH} : \left[\begin{array}{rcl} \textsf{in} & : & \emptyset \\ \textsf{out} & : & \{\textsf{prop}\} \\ \textsf{link} & : & \{\textsf{prop} \mapsto \{\textsf{vinf}\}\} \end{array} \right] \right]
$$

Overview

Syntax-semantics interface to CLLS

- now, we can make use of the information contained in the TDGS analyses for creating the interface to CLLS

Syntax-semantics interface to CLLS

- now, we can make use of the information contained in the TDGS analyses for creating the interface to CLLS
- we construct underspecified semantics in the Constraint Language for Lambda Structures (CLLS, Egg et al 01)

CLLS

- CLLS: constraint language talking about *lambda structures*

CLLS

- CLLS: constraint language talking about *lambda structures*
- a lambda structure represents a higher-order λ-term in a graph

Example: weak reading

- weak reading of “Jeder Mann liebt eine Frau”: λ-term:

Example: weak reading

- weak reading of “Jeder Mann liebt eine Frau”: λ-term:

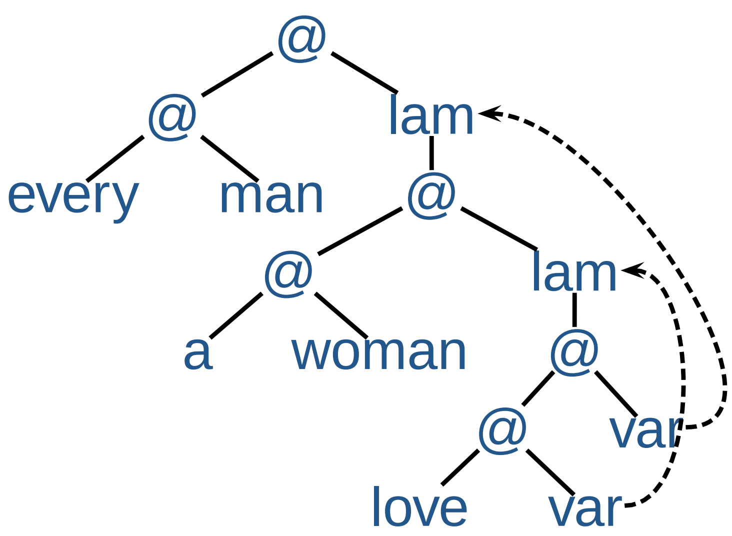
$$
\begin{aligned}
&(\text{every man})(\lambda x. \\
&\quad (\text{a woman})(\lambda y. \\
&\qquad (\text{love } y)\ x))
\end{aligned}
$$

Example: weak reading

- weak reading of “Jeder Mann liebt eine Frau”: λ-term:

$$\begin{array}{l}(\text{every man})(\lambda x.\\ \quad(\text{a woman})(\lambda y.\\ \quad\quad(\text{love } y)\ x))\end{array}$$

- CLLS lambda structure:



Example: strong reading

- strong reading of “Jeder Mann liebt eine Frau”: λ-term:

Example: strong reading

- strong reading of “Jeder Mann liebt eine Frau”: λ-term:

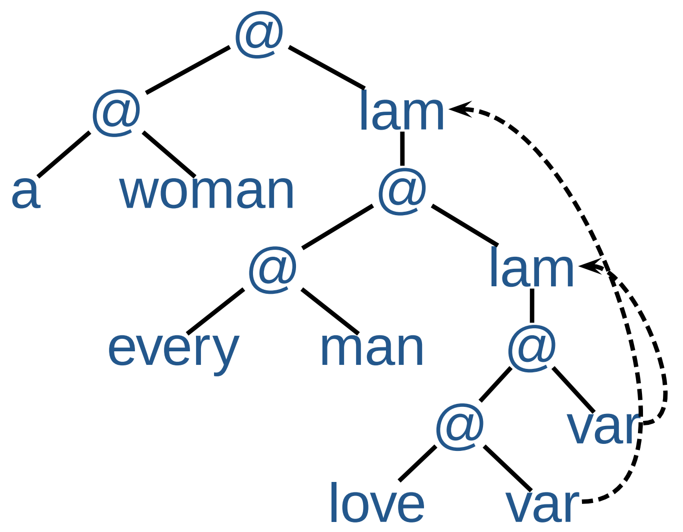
$$\begin{aligned}&(\text{a woman})(\lambda y.\\&\quad(\text{every man})(\lambda x.\\&\qquad(\text{love } y)\ x))\end{aligned}$$

Example: strong reading

- strong reading of “Jeder Mann liebt eine Frau”: λ-term:

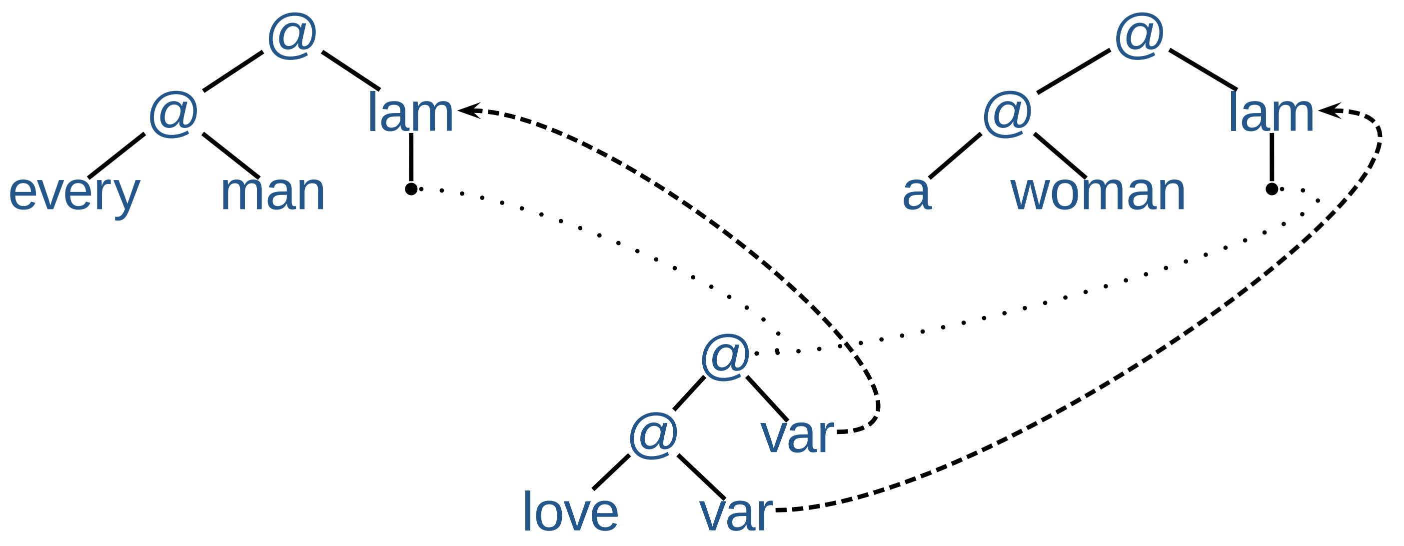
$$\begin{aligned}&(\text{a woman})(\lambda y.\\ &\quad(\text{every man})(\lambda x.\\ &\qquad(\text{love } y)\ x))\end{aligned}$$

- CLLS lambda structure:



Underspecifi cation

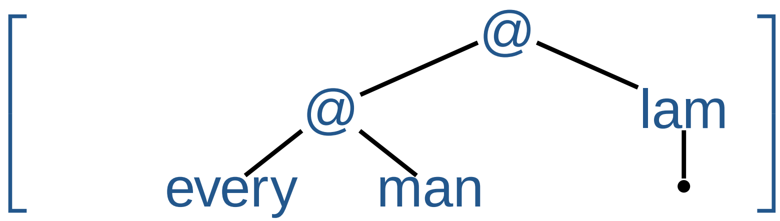
- encodes the weak and strong readings in one CLLS constraint:



Syntax-semantics interface: nouns

- in the lexicon, assign CLLS fragments to words, e.g.:

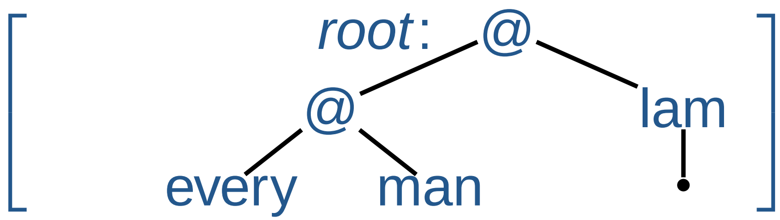
$jeder\ Mann \mapsto$



Syntax-semantics interface: nouns

- identify position of root:

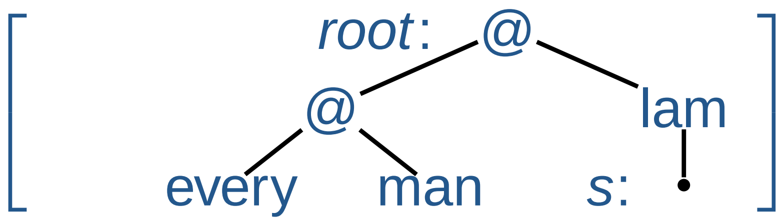
$jeder\ Mann \mapsto$



Syntax-semantics interface: nouns

- identify position of scope:

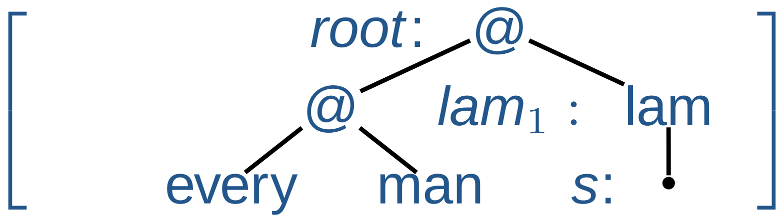
$$jeder\ Mann \mapsto$$



Syntax-semantics interface: nouns

- identify position of the lambda binder:

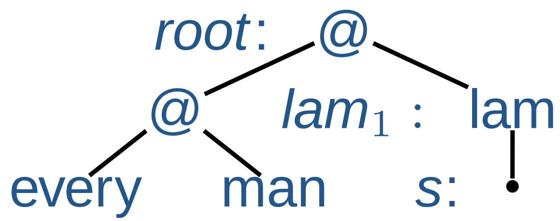
$jeder\ Mann \mapsto$



Syntax-semantics interface: nouns

- establish mapping from TH edge labels to lambda binders:

$jeder\ Mann \mapsto$

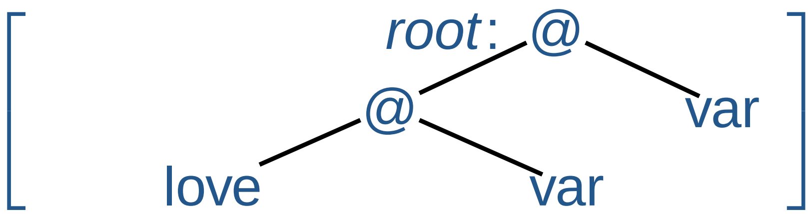


$$\left[\ \text{lam} : \begin{bmatrix} \text{act} & : & \mathit{lam}_1 \\ \text{pat} & : & \mathit{lam}_1 \end{bmatrix}\ \right]$$

Syntax-semantics interface: verbs

- also identify root node:

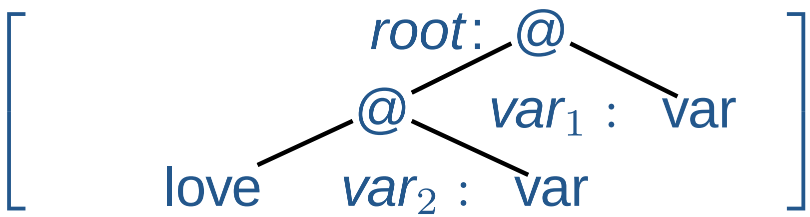
zu lieben $\mapsto$



Syntax-semantics interface: verbs

- identify variable positions:

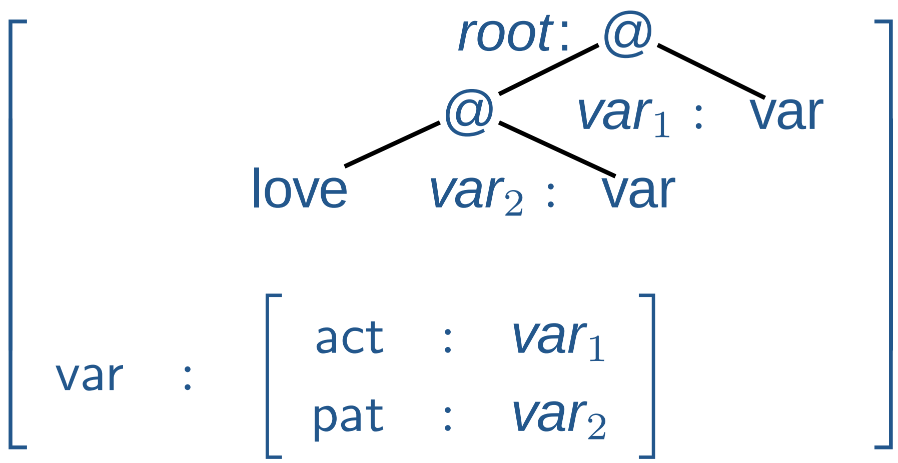
$zu\ lieben \mapsto$



Syntax-semantics interface: verbs

- and establish a mapping from TH edge labels to variable positions:

zu lieben $\mapsto$



Meaning assembly

- finally, use the information contained in the TH dag to get the CLLS constraints:

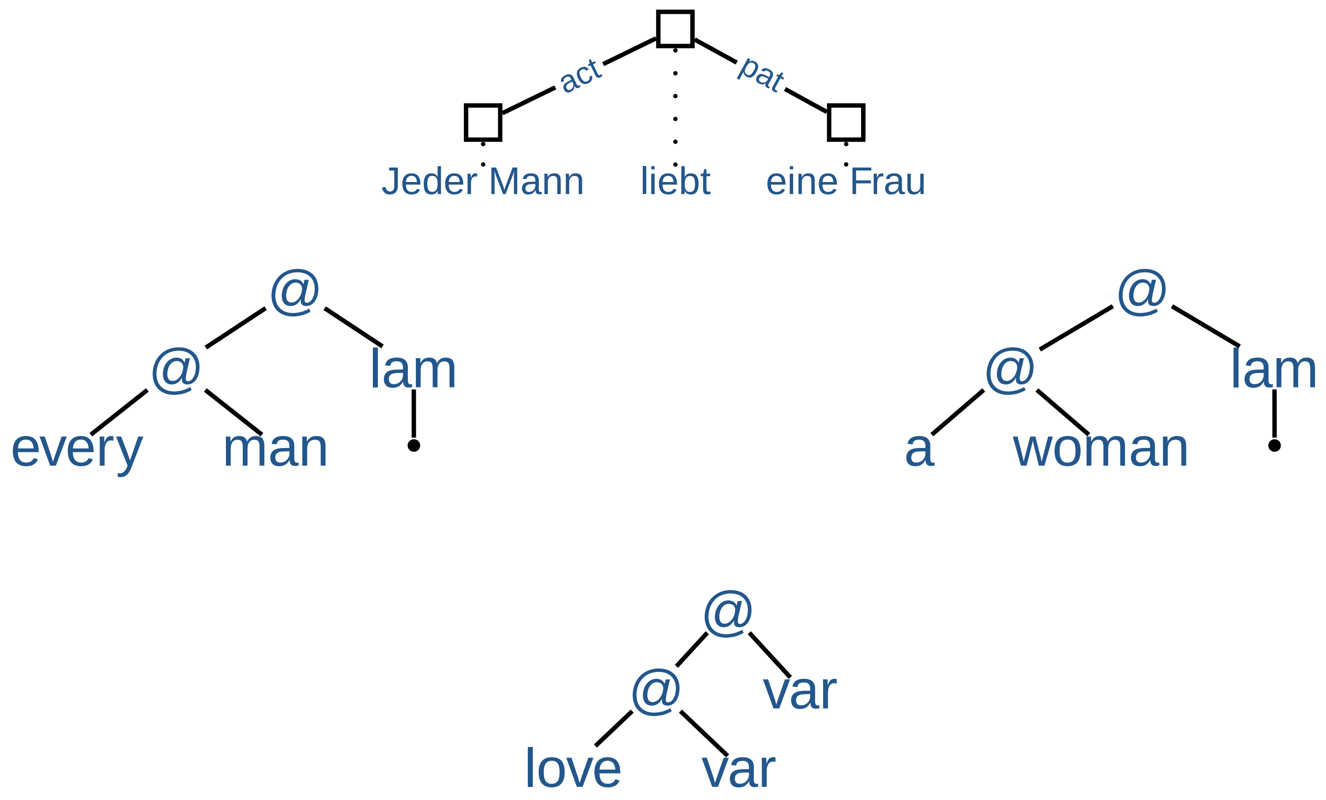
Meaning assembly

- finally, use the information contained in the TH dag to get the CLLS constraints:

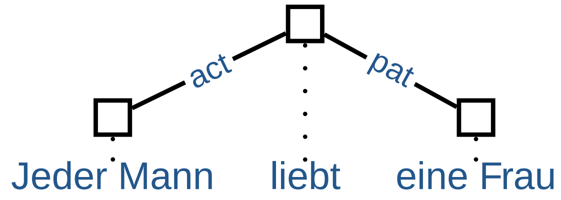
$$v{-}\ell{\rightarrow}_{\mathsf{TH}} v' \quad \Rightarrow \quad v.\mathsf{var}.\ell \dashrightarrow v'.\mathsf{lam}.\ell \quad \wedge$$

$$v'.\mathit{s} \; \cdots\rightarrow \; v.\mathit{root}$$

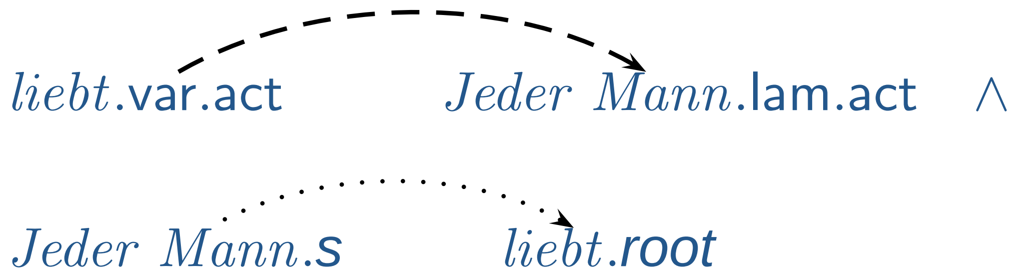
Example


act
pat
Jeder Mann
liebt
eine Frau
@
@
lam
every
man
@
@
lam
a
woman
@
@
var
love
var

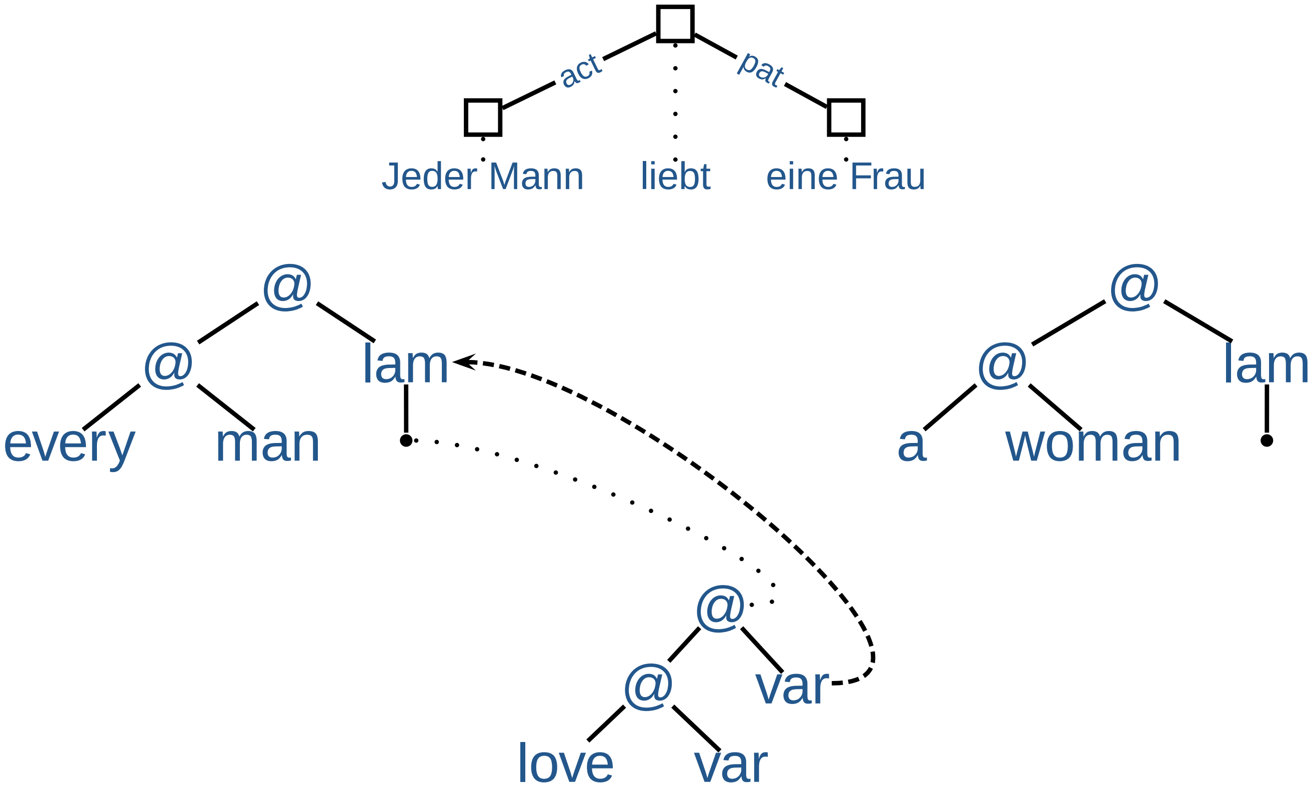
Example



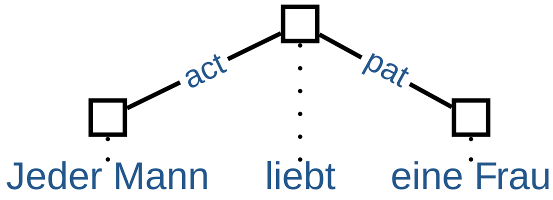
$$\mathit{liebt} \xrightarrow{\mathsf{act}}_{\mathsf{TH}} \mathit{Jeder\ Mann} \Rightarrow$$



Example


act
pat
Jeder Mann
liebt
eine Frau
@
@
lam
every
man
@
@
lam
a
woman
@
@
var
love
var

Example

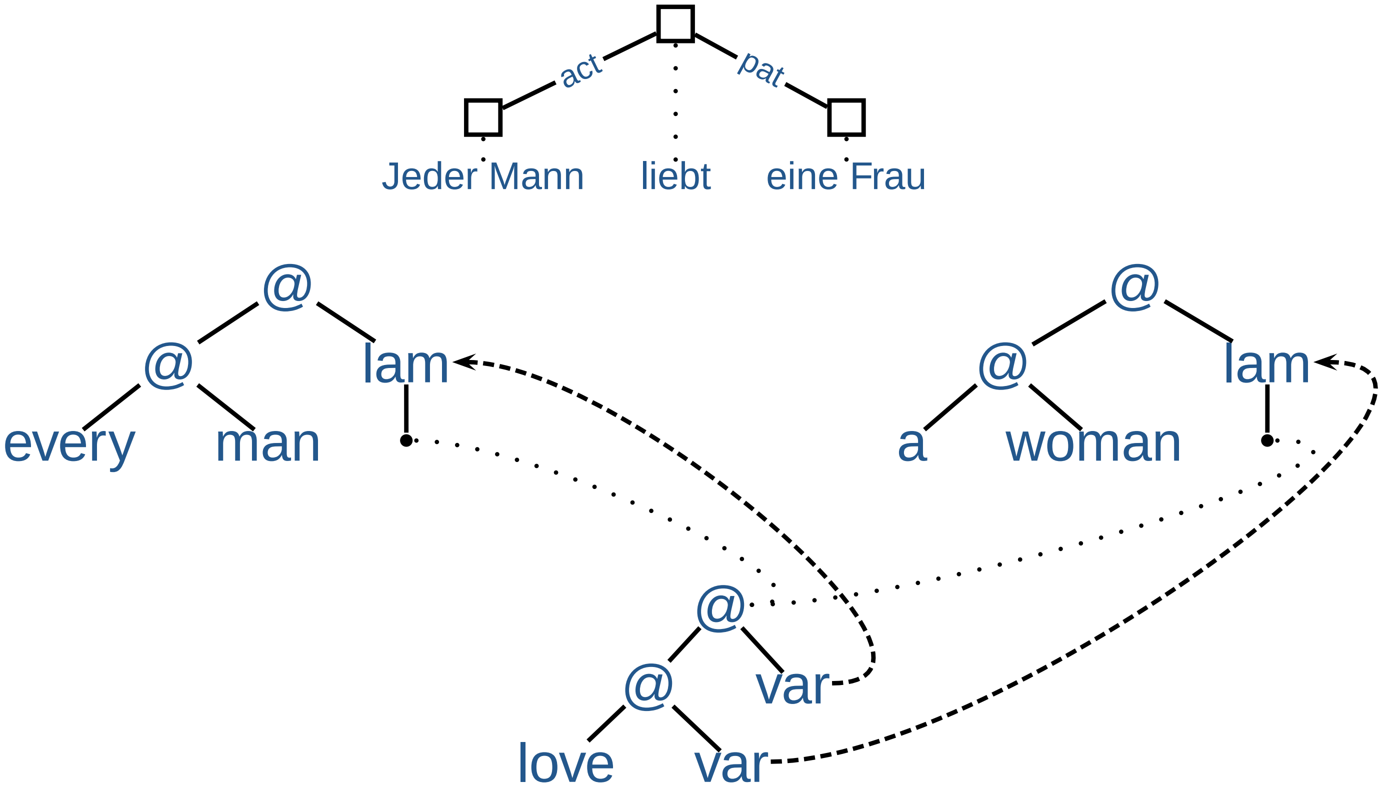


$$liebt \text{—pat}\!\rightarrow_{\text{TH}} eine\ Frau \Rightarrow$$

$$liebt.\text{var.pat} \qquad eine\ Frau.\text{lam.pat} \qquad \wedge$$

$$eine\ Frau.\text{s} \qquad liebt.\textit{root}$$

Example



act
pat
Jeder Mann
liebt
eine Frau
@
@
lam
every
man
@
@
lam
a
woman
@
@
var
love
var

Overview

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism
- XDG is a generalization of Topological Dependency Grammar (TDG)

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism
- XDG is a generalization of Topological Dependency Grammar (TDG)
- XDG can be instantiated to yield particular grammar formalisms, e.g. TDG, and TDGS

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism
- XDG is a generalization of Topological Dependency Grammar (TDG)
- XDG can be instantiated to yield particular grammar formalisms, e.g. TDG, and TDGS
- TDGS includes a TH dimension to reflect semantic argument structure

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism
- XDG is a generalization of Topological Dependency Grammar (TDG)
- XDG can be instantiated to yield particular grammar formalisms, e.g. TDG, and TDGS
- TDGS includes a TH dimension to reflect semantic argument structure
- the TH information can be used by a syntax-semantics interface to construct underspecified CLLS semantics

Conclusion

- introduced eXtensible Dependency Grammar (XDG) meta grammar formalism
- XDG is a generalization of Topological Dependency Grammar (TDG)
- XDG can be instantiated to yield particular grammar formalisms, e.g. TDG, and TDGS
- TDGS includes a TH dimension to reflect semantic argument structure
- the TH information can be used by a syntax-semantics interface to construct underspecified CLLS semantics