XDG - A Metagrammatical Framework for Dependency Grammar

Ralph Debusmann

Programming Systems Lab

Universität des Saarlandes

This talk

- introduces a new metagrammatical framework for dependency grammar: eXtensible Dependency Grammar (XDG)
- evolved as a generalisation of Topological Dependency Grammar (TDG) (Duchier and Debusmann 2001)
- metagrammatical: can be instantiated to yield specific grammar formalisms (including TDG itself)
- based on dependency grammar

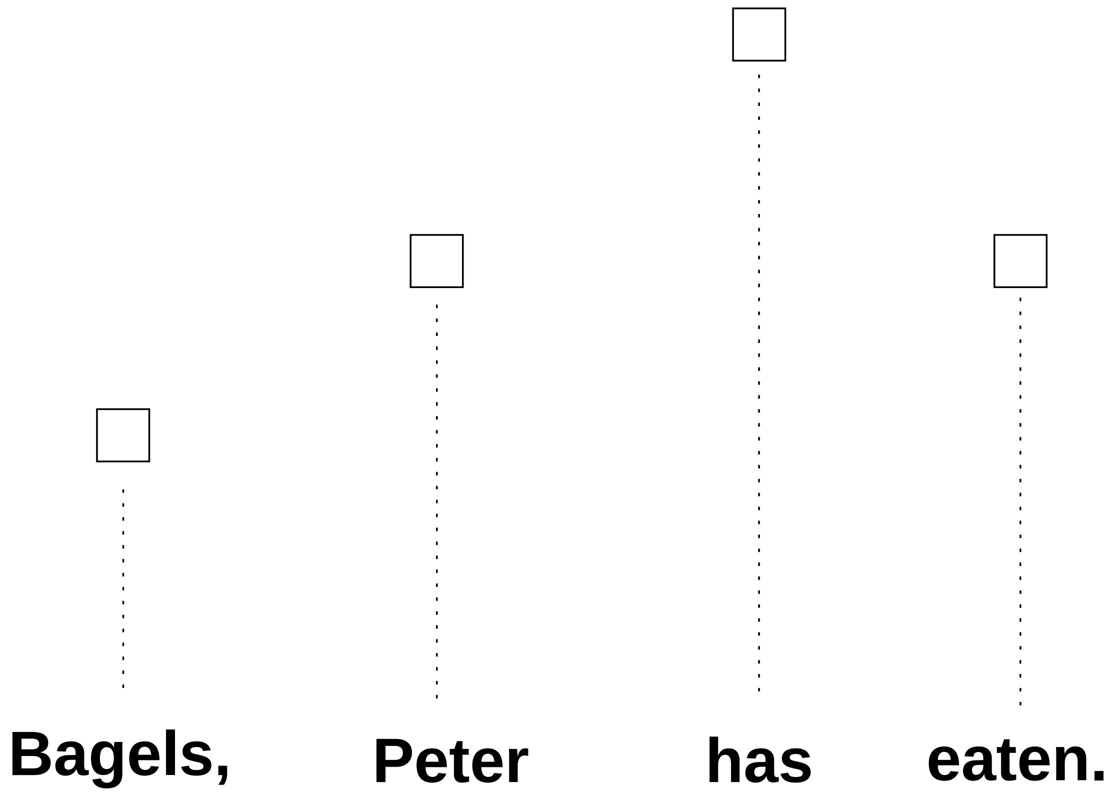
Dependency grammar

- collection of ideas for natural language analysis
- long history (following Kruijff 2002):
  - Greek and Latin scholars: Thrax, Apollonius, and Priscian
  - Indian: Panini's formal grammar of Sanskrit (Astadhyayi/Astaka, 350/250 BC)
  - Arabic: Kitab al-Usul of Ibn al-Sarrag (d.928)
  - European: Martinus de Dacia (d.1304), Thomas von Erfurt (14th century)
- modern dependency grammar credited to Tesniere (1959)
- so what are these ideas?

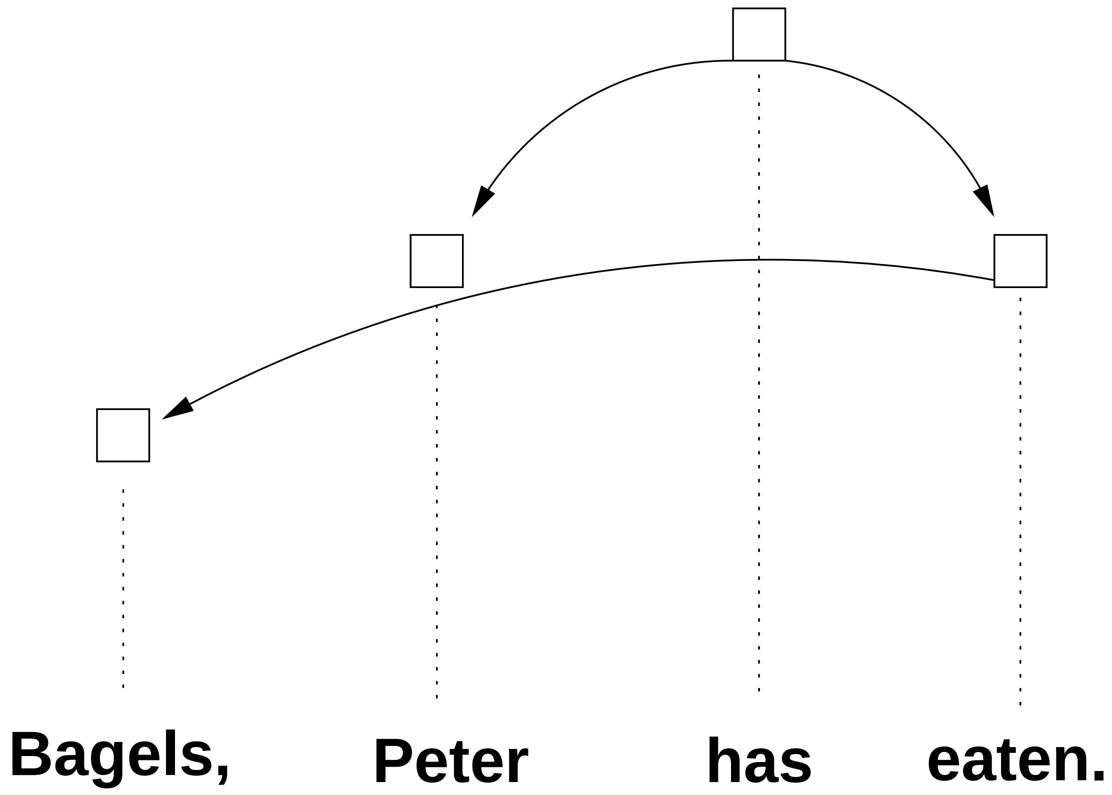
Words

Bagels, Peter has eaten.

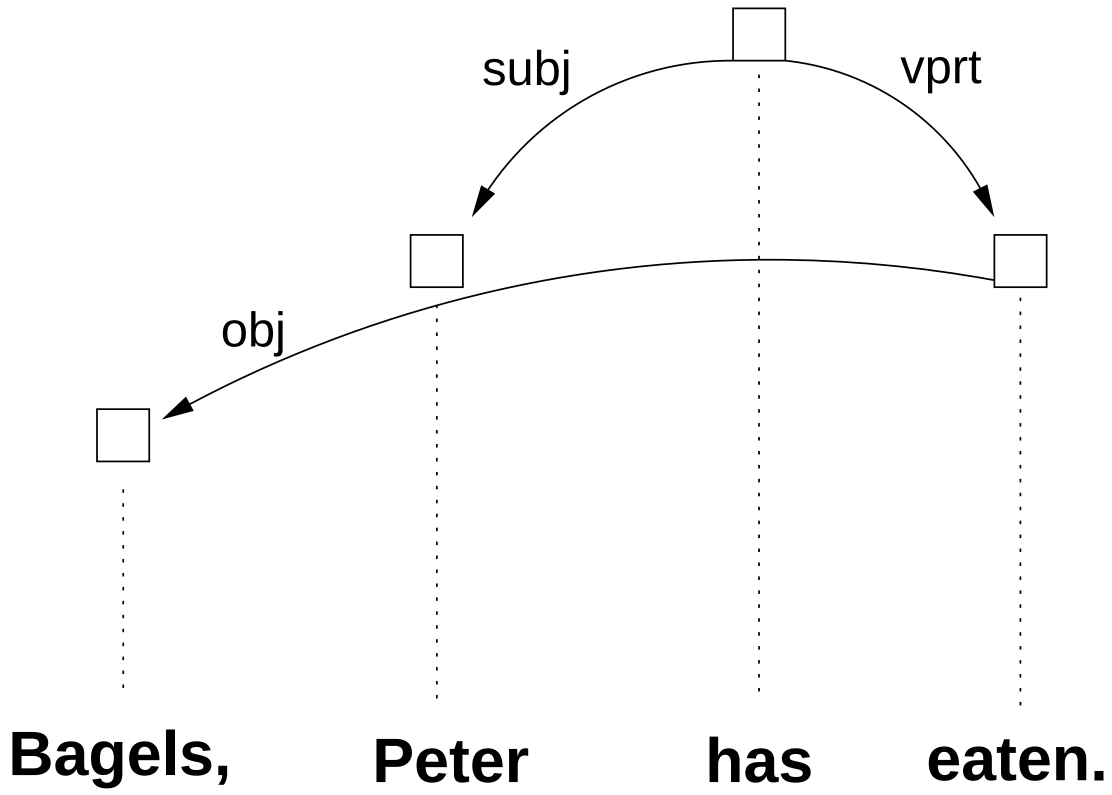
1:1-correspondence between words and nodes



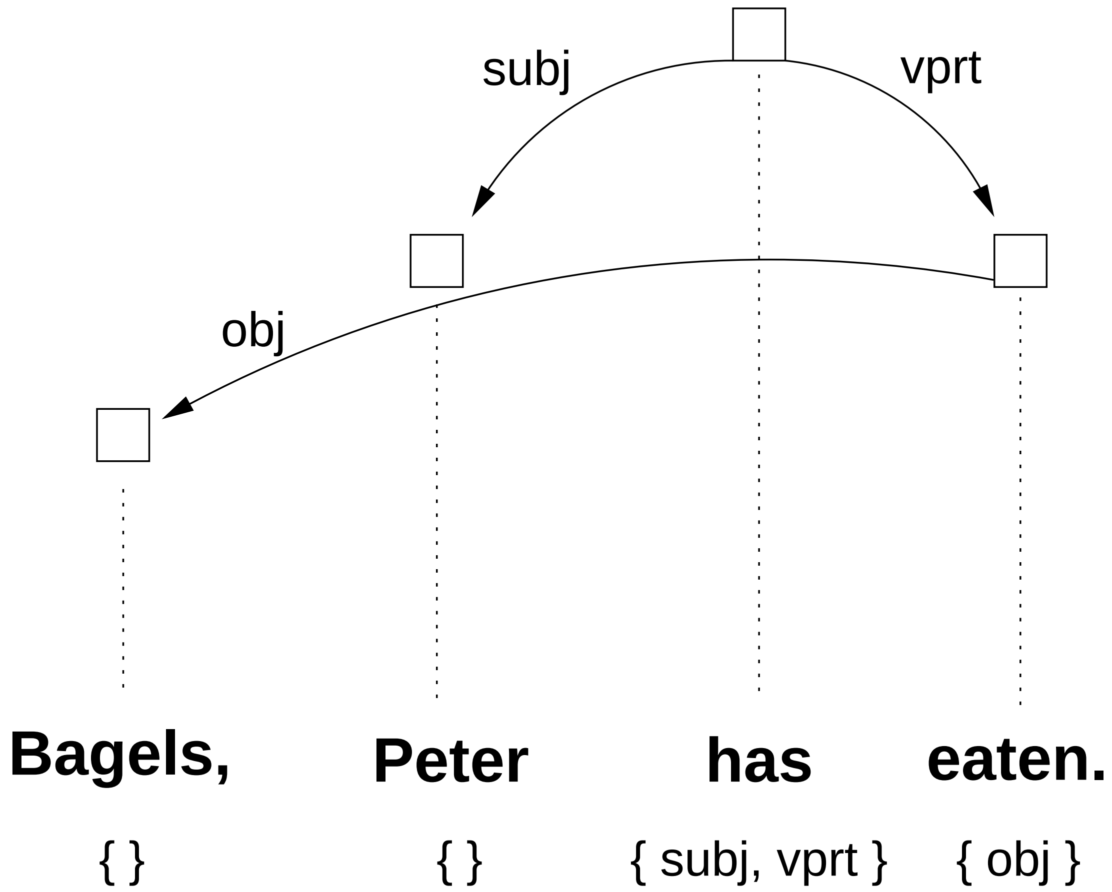
Head/dependent-asymmetry



Grammatical functions (edge labels)



Valency (subcategorisation)



Dependency and phrase structure

- ideas from dependency grammar adopted by many phrase structure-based grammar formalisms:
  - Government and Binding (GB, Chomsky 1986): X’-theory
  - Head-driven Phrase Structure Grammar (HPSG, Pollard and Sag 1994): e.g. DEPS-feature in modern variants (Bouma, Malouf and Sag 1998)
  - Lexical Functional Grammar (LFG, Bresnan and Kaplan 1982): f-structure
  - Tree Adjoining Grammar (TAG, Joshi 1987): derivation tree

Pure dependency grammar formalisms

- pure dependency grammar formalisms have been less successful:
  - Abhaengigkeitsgrammatik (Kunze 1975)
  - Functional Generative Description (FGD, Sgall et al 1986)
  - Meaning Text Theory (MTT, Melcuk 1988)
  - Word Grammar (Hudson 1990)
- why?

Problems of pure dependency grammar formalisms

- word order: no declarative specification
- syntax-semantics interface: no compositional semantics construction

Word order

- MSc thesis (Debusmann 2001): TDG grammar formalism
- allows declarative specification of word order
- constraint-based parser for TDG (Duchier 1999)
- parser average case efficient (but only small test grammars), although TDG parsing is NP-complete (Koller and Striegnitz 2002)
- TDG parser used for LTAG generation by Koller and Striegnitz 2002, faster than the generator described in Carrol et al 1999
- (Kuhlmann 2002): TDG parser used for parsing Categorial Type Logics (CTL) (Morrill 1994, Moortgat 1997)

Syntax-semantics interface

- goal of PhD research: develop a syntax-semantics interface for dependency grammar
- idea:
  1. generalise TDG into a metagrammatical framework for dependency grammar (XDG)
  2. use XDG to develop the syntax-semantics interface

Roadmap of the talk

1. XDG
   - basic architecture
   - principles (stipulating the well-formedness conditions of analyses)
   - lexicalisation
2. TDG as an instance of XDG
3. syntax-semantics interface
   - Semantic Topological Dependency Grammar (STDG)
   - STDG as another instance of XDG
4. conclusions

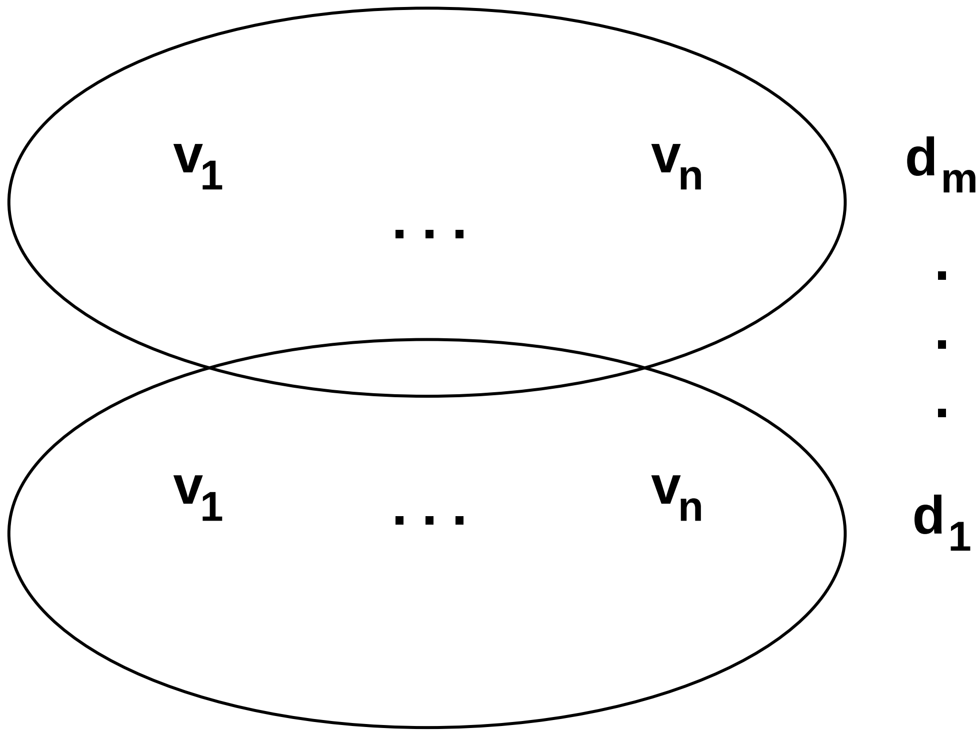
eXtensible Dependency Grammar (XDG)

- graph description language
- describes a set of *graph dimensions*
- a graph dimension is a labeled directed graph $G_d(V, E_d)$
- all graph dimensions share the same set V of nodes
- each graph dimension has its own set E_d of labeled edges (L_d set of edge labels, $E_d \subseteq V \times L_d \times V$)
- simple feature structures can be attached to each node (features: functions $V \longrightarrow R$, where R is an arbitrary codomain)
- parametrised *principles* stipulate well-formedness conditions

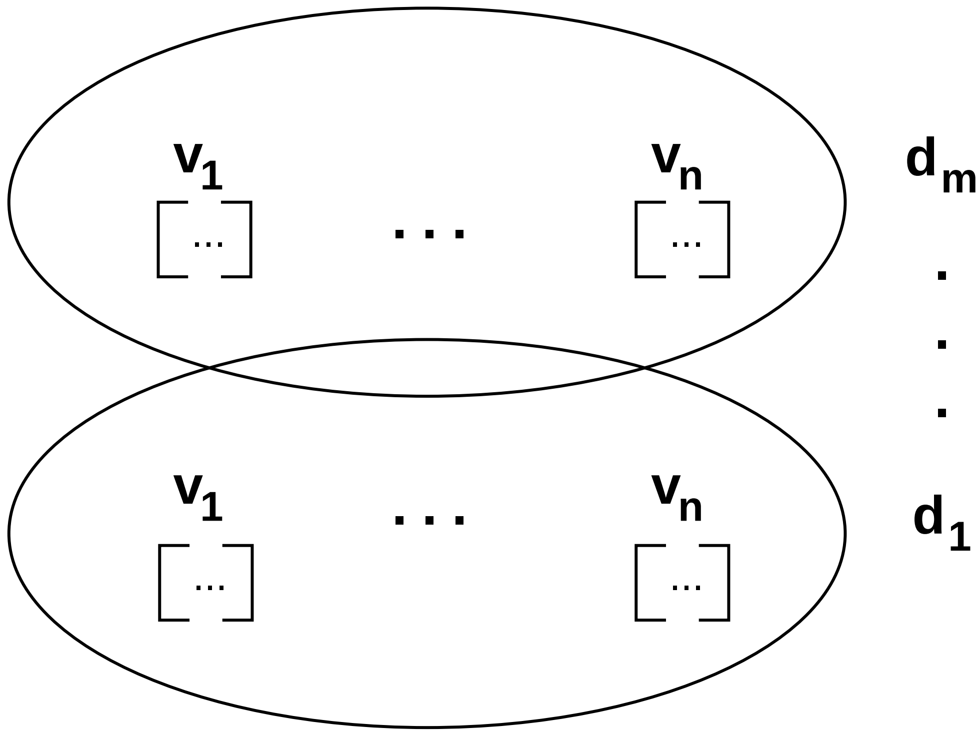
Nodes (arranged in a graph)

v_1 . . . v_n

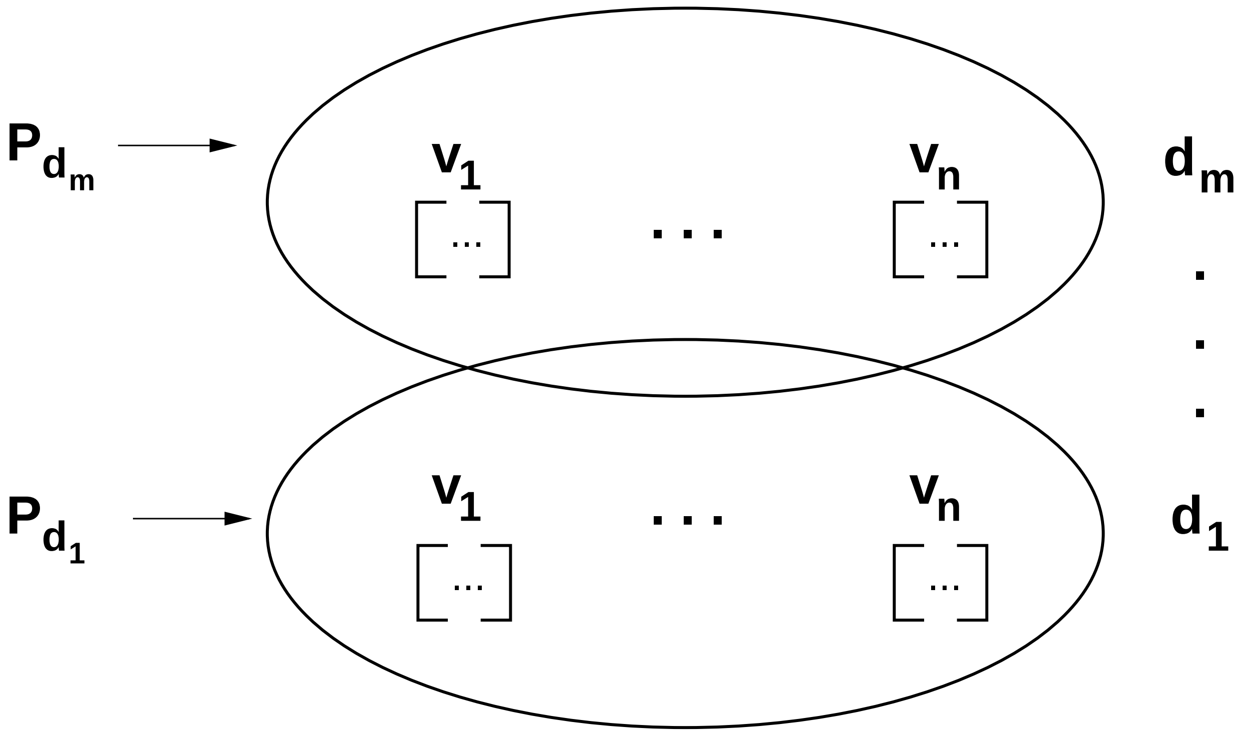
Graph dimensions



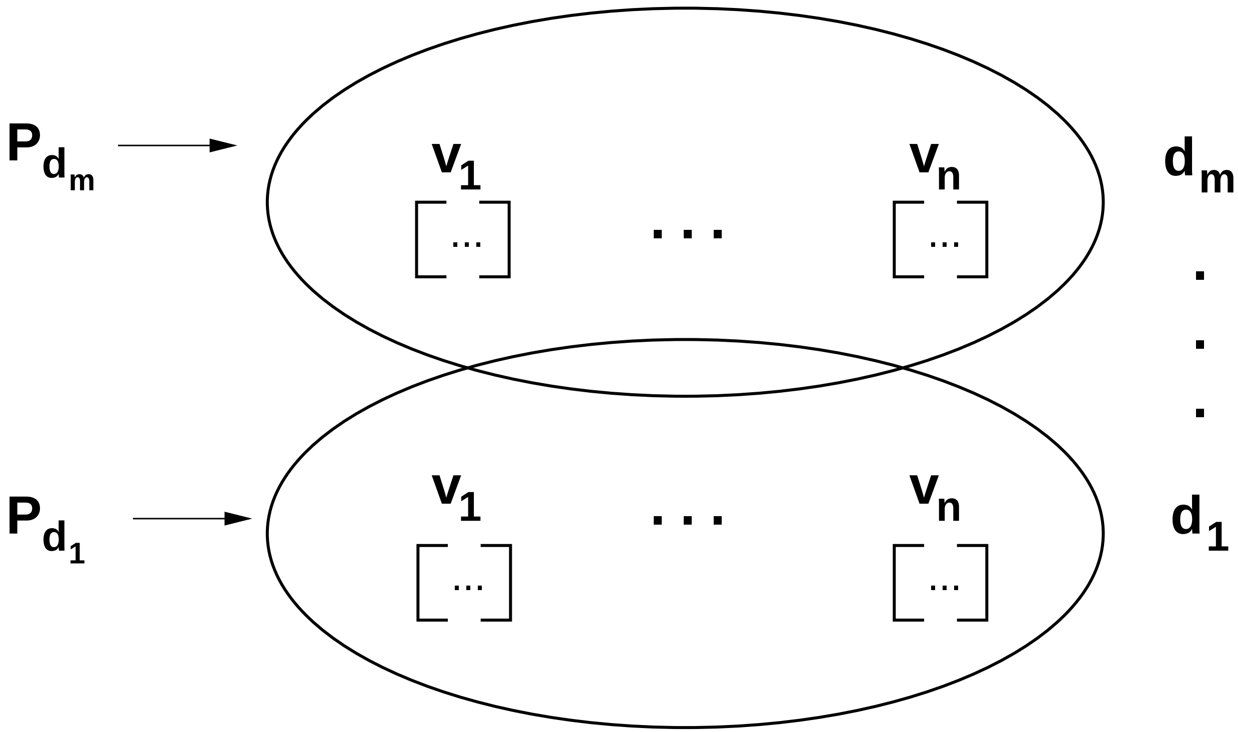
Feature structures



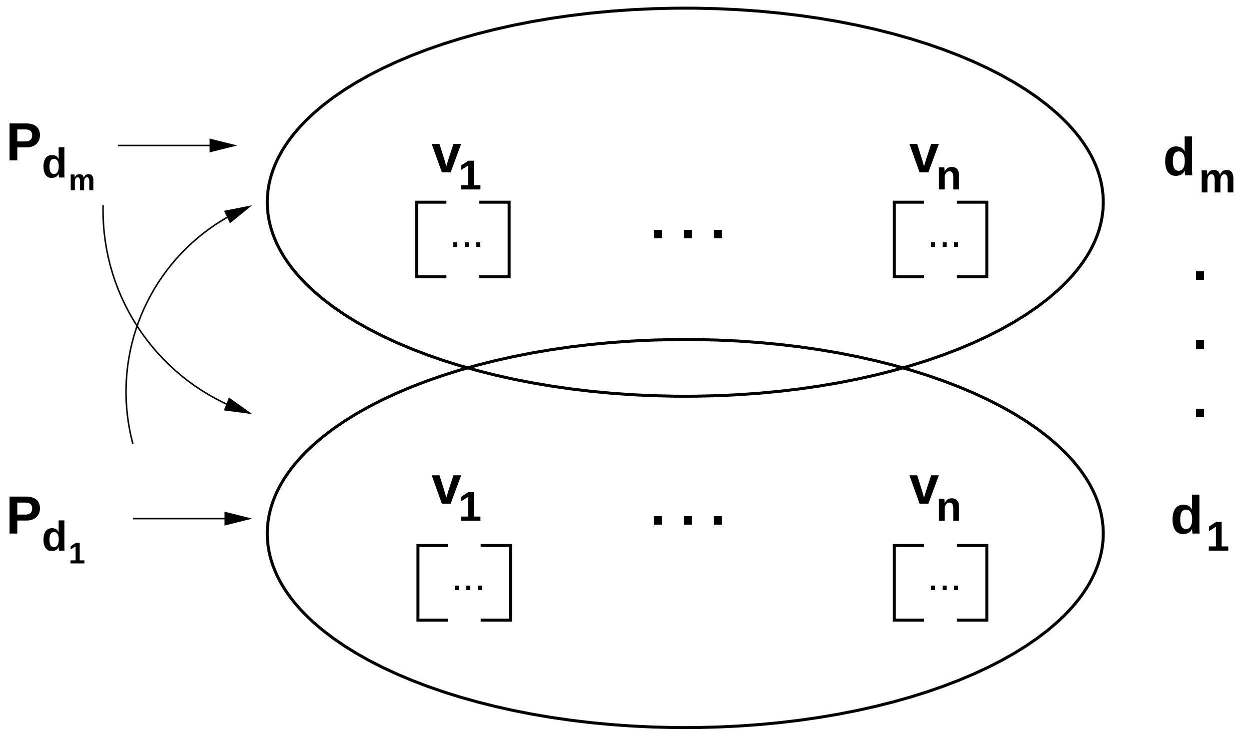
Principles



Principles (one-dimensional)



Principles (multi-dimensional)



Principle library

- directed acyclic graph *
- tree *
- in
- out *
- order
- projectivity
- climbing
- barriers
- linking *
- covariance *
- contravariance *
- node and edge constraints
- ... (extensible)

Directed acyclic graph

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

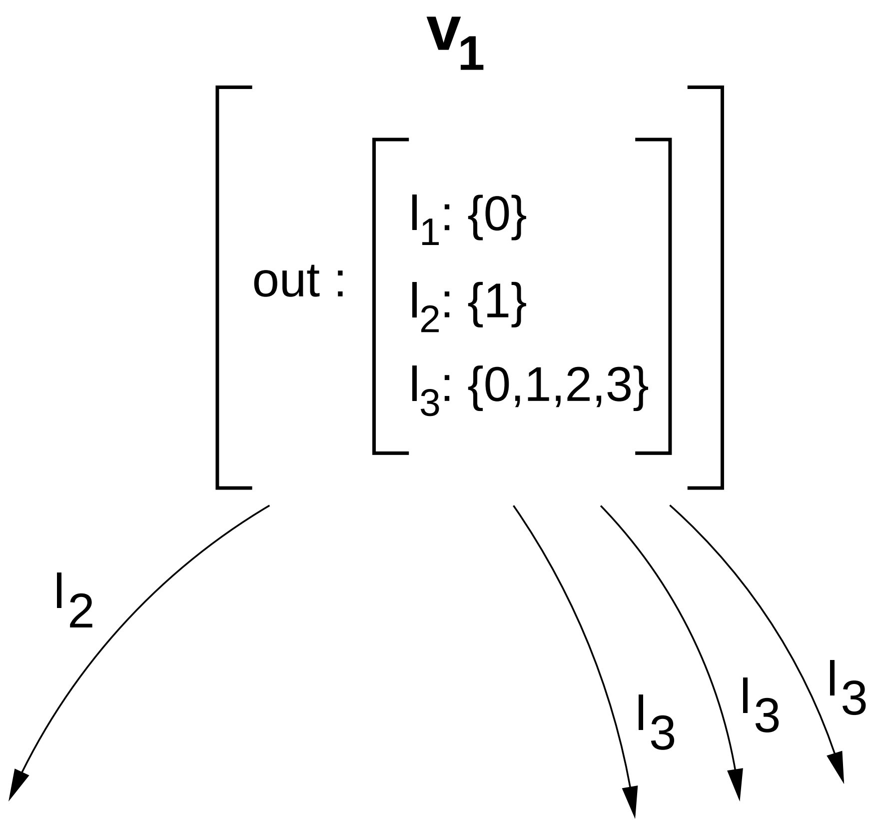
Tree

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

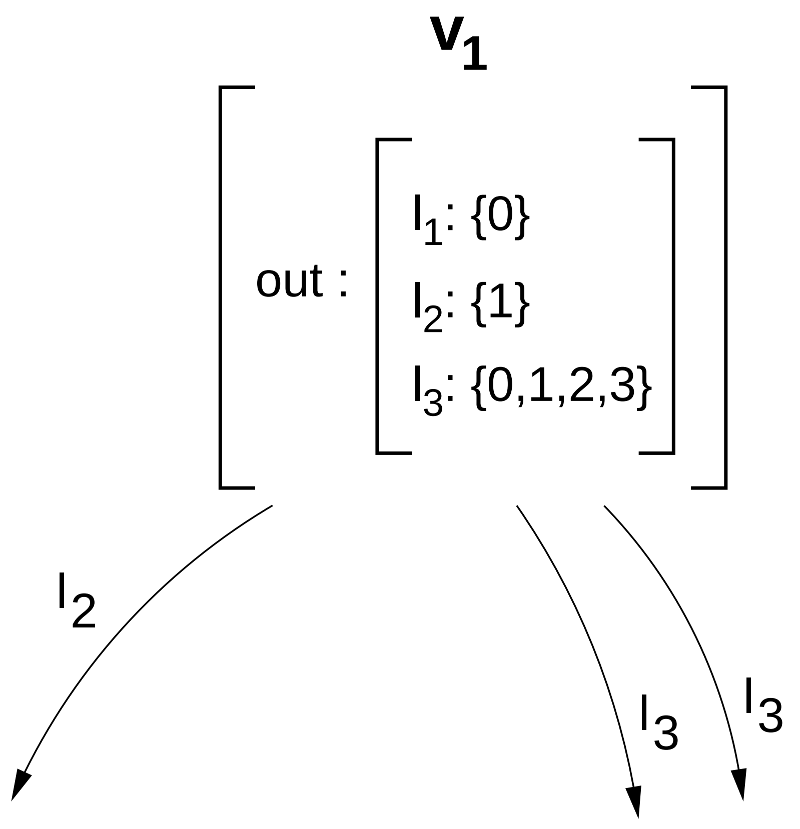
Out

$\mathtt{out}(G_d, \mathsf{f})$: The outgoing edges of each node in G_d must satisfy the nodes' *out specification*. Feature $\mathsf{f} : V \to (L_d \to 2^{\mathbb{N}})$ maps an out specification to each node.

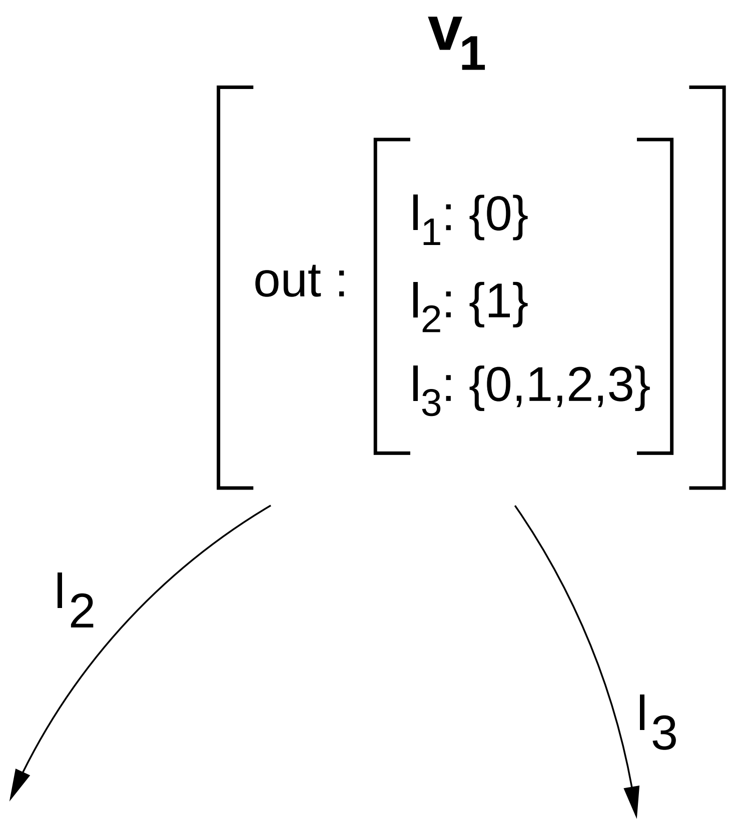
Out



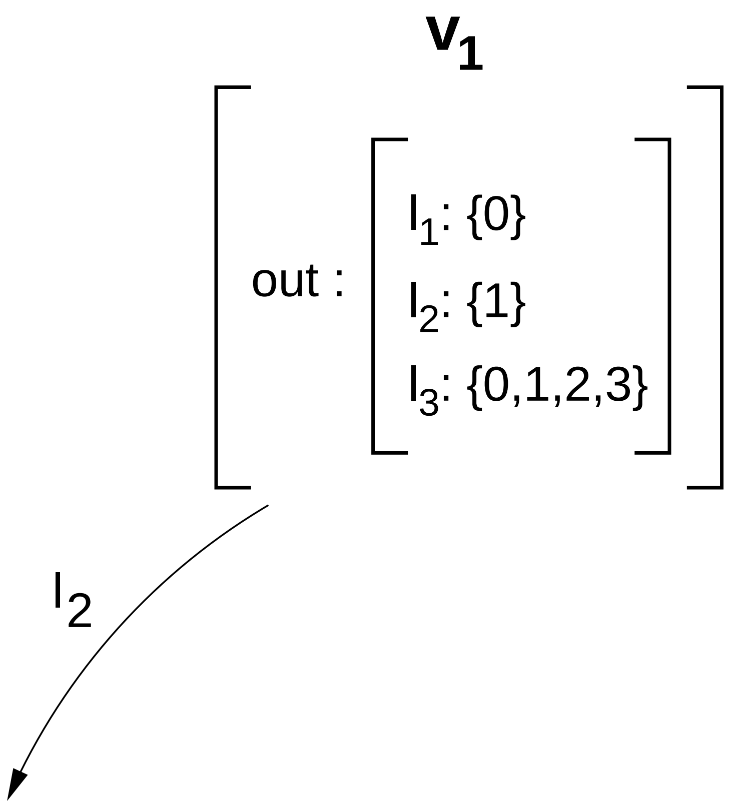
Out



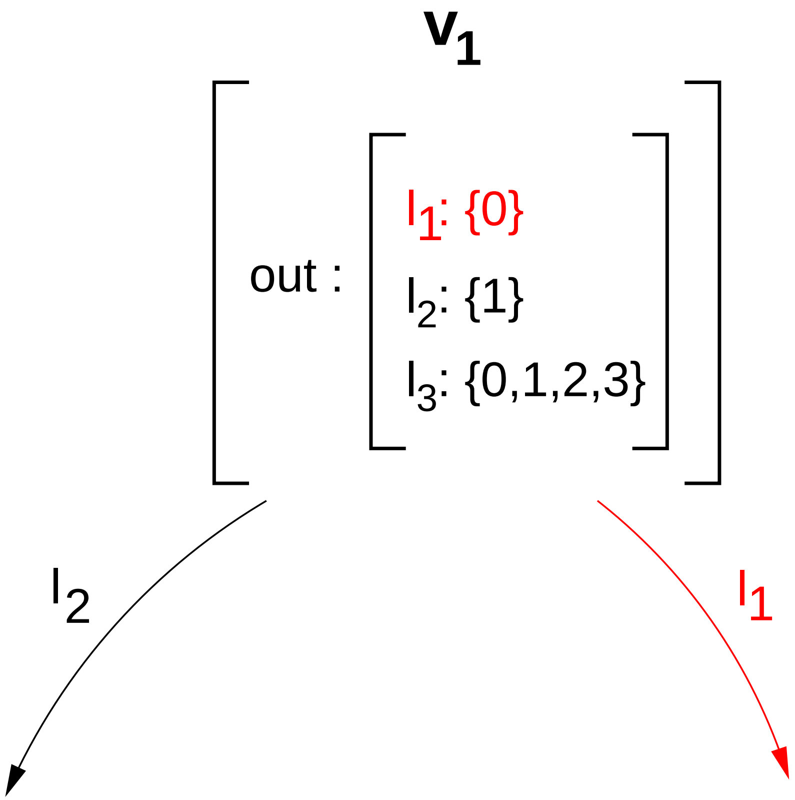
Out



Out



Out



Out

$$v_1 \left[\text{out} : \begin{bmatrix} l_1 : \{0\} \\ l_2 : \{1\} \\ l_3 : \{0,1,2,3\} \end{bmatrix} \right]$$

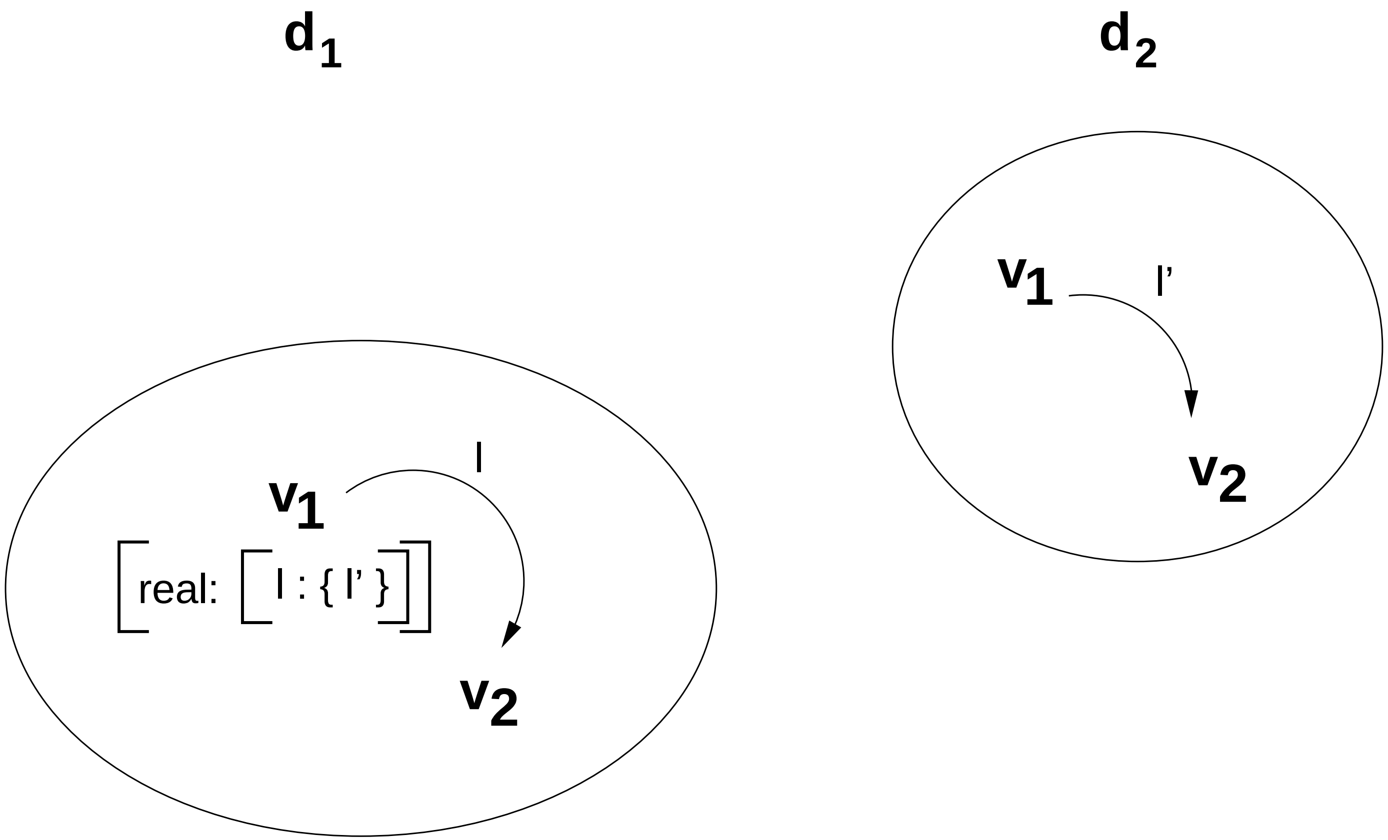
Linking

$\mathsf{linking}(G_{d_1}, G_{d_2}, \mathsf{f}_1, \mathsf{f}_2)$: An edge (v_1, l, v_2) in G_{d_1} is only licensed if v_1 *realises* l by $\mathsf{l}' \in L_{d_2}$, and either:

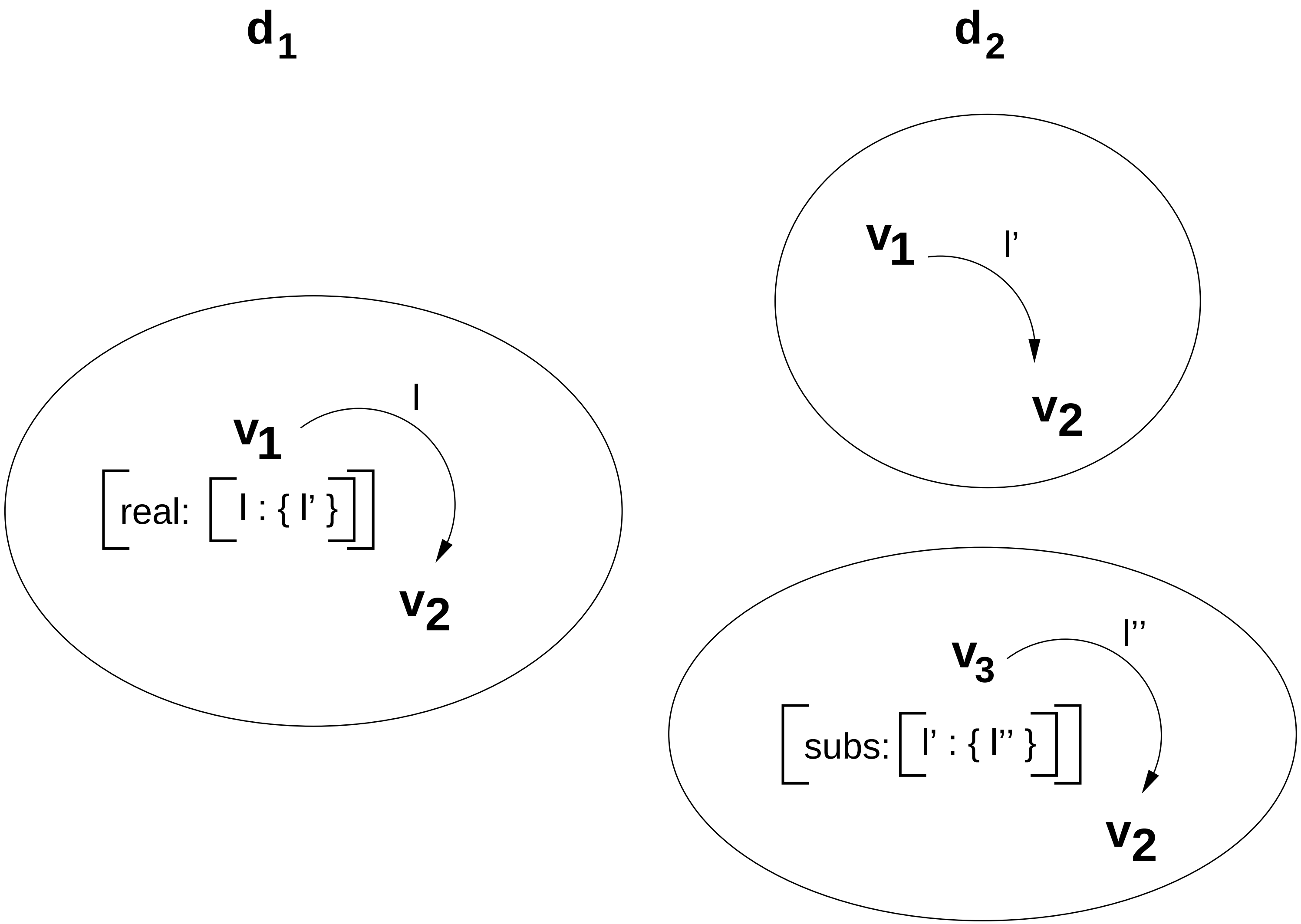
1. there is a corresponding edge (v_1, l', v_2) in G_{d_2}, or
2. there is an edge (v_3, l'', v_2) in G_{d_2} and v_3 *substitutes* l' by l''.

Feature $\mathsf{f}_1 : V \to (L_{d_1} \to 2^{L_{d_2}})$ assigns to each node a label realisation function. Feature $\mathsf{f}_2 : V \to (L_{d_2} \to 2^{L_{d_2}})$ assigns to each node a label substitution function.

Linking



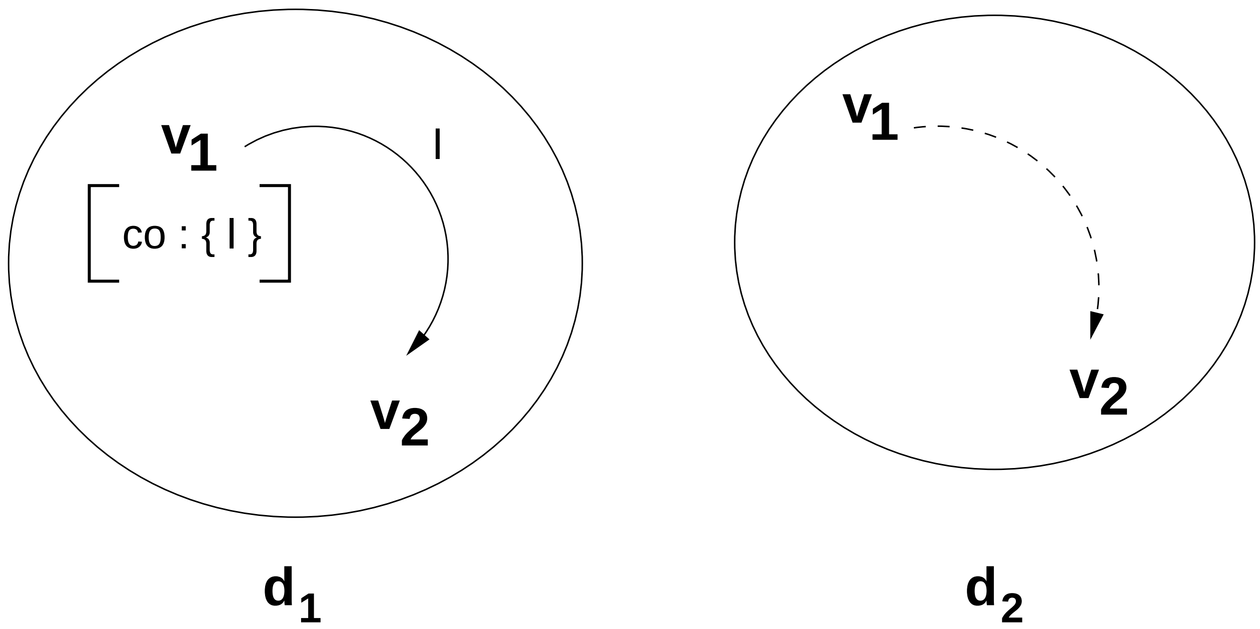
Linking



Covariance

$\mathsf{covariance}(G_{d_1}, G_{d_2}, \mathsf{f})$: Each edge (v_1, l, v_2) in G_{d_1} where l is *covariant* on v_1 is only licensed if v_1 is above v_2 in G_{d_2}. Feature $\mathsf{f} : V \to 2^{L_{d_1}}$ assigns to each node its set of covariant labels.

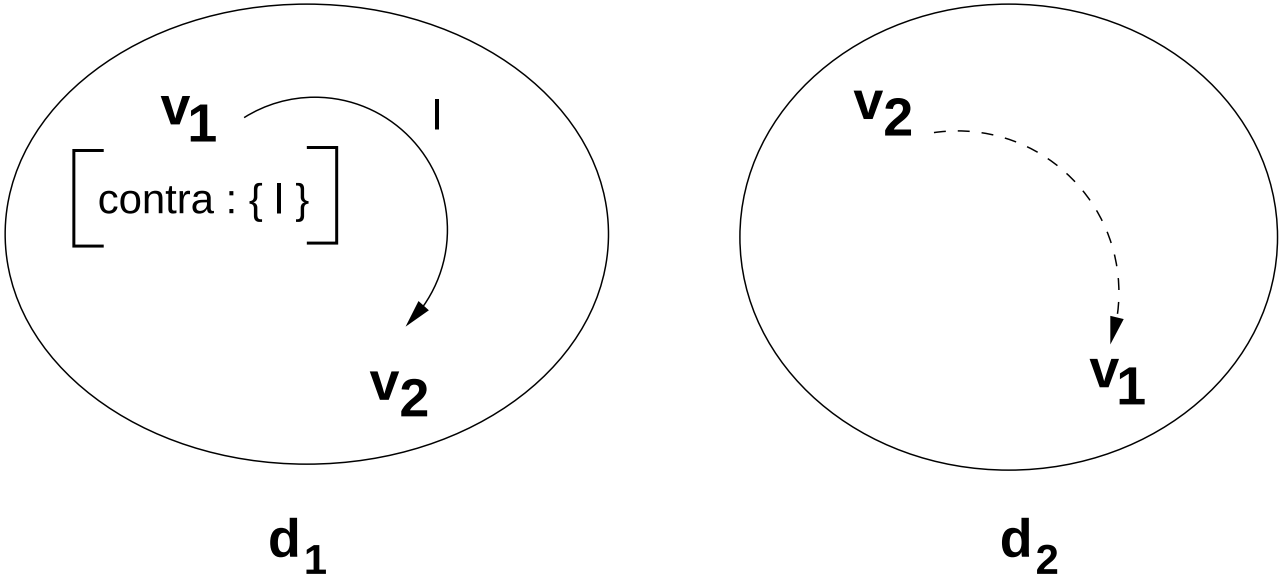
Covariance



Contravariance

contravariance$(G_{d_1}, G_{d_2}, \mathsf{f})$: Each edge (v_1, l, v_2) in G_{d_1} where l is *contravariant* on v_1 is only licensed if v_1 is below v_2 in G_{d_2}. Feature $\mathsf{f} : V \to 2^{L_{d_1}}$ assigns to each node its set of contravariant labels.

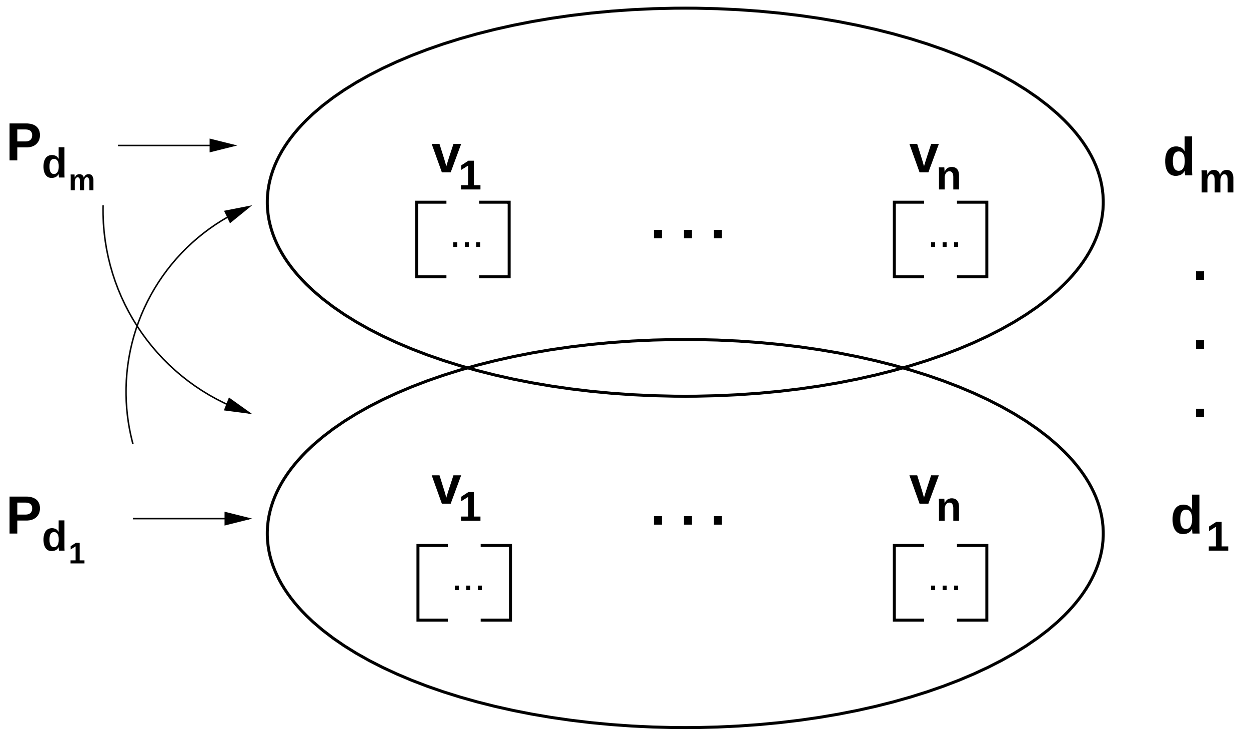
Contravariance



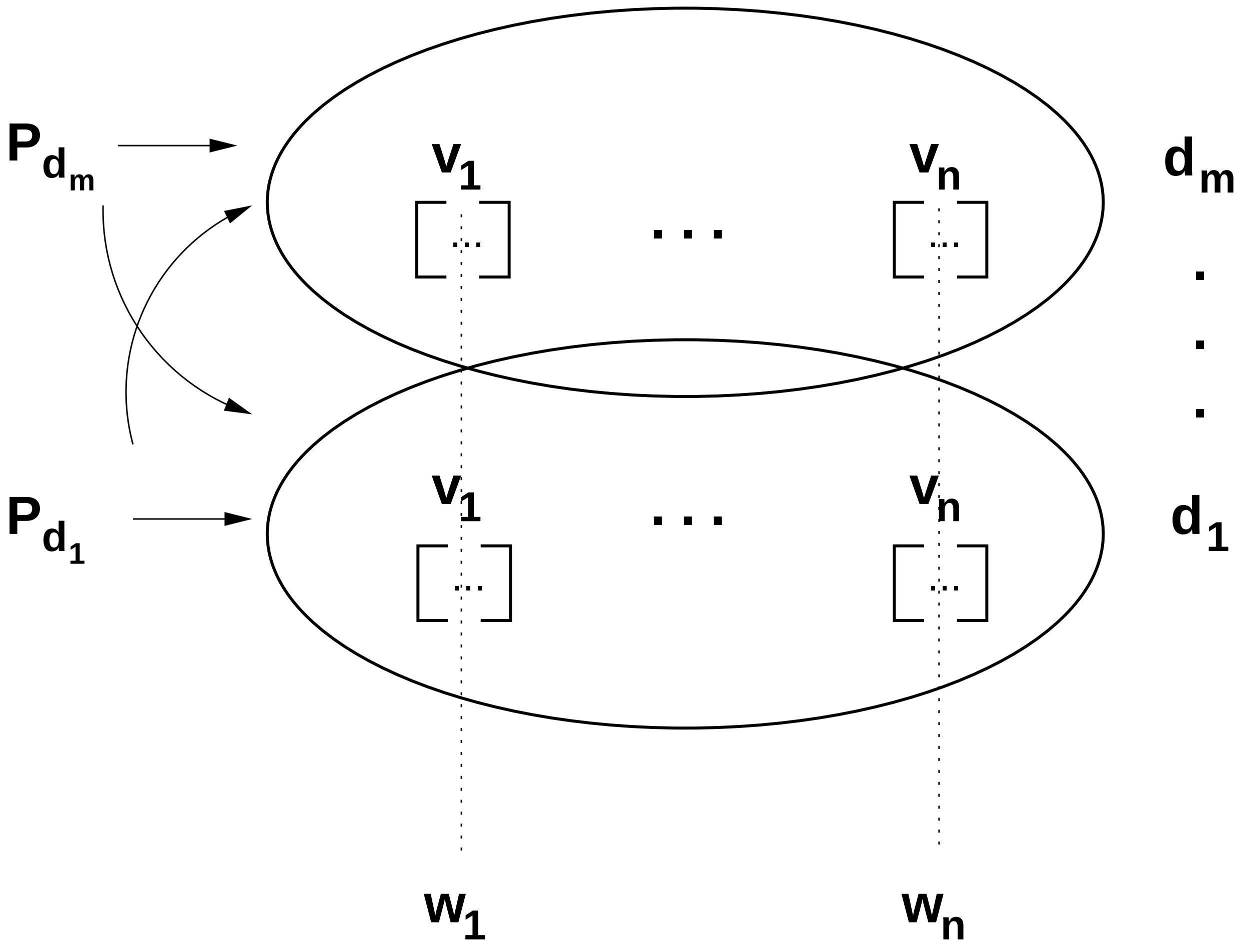
Lexicalisation

1. from dependency grammar: 1:1-correspondence between nodes and words
2. assign to each word a set of lexical entries (feature structures)
3. select one of the lexical entries, efficient through selection constraint (Duchier 1999)
4. assign the selected entry (feature structure) to the corresponding node

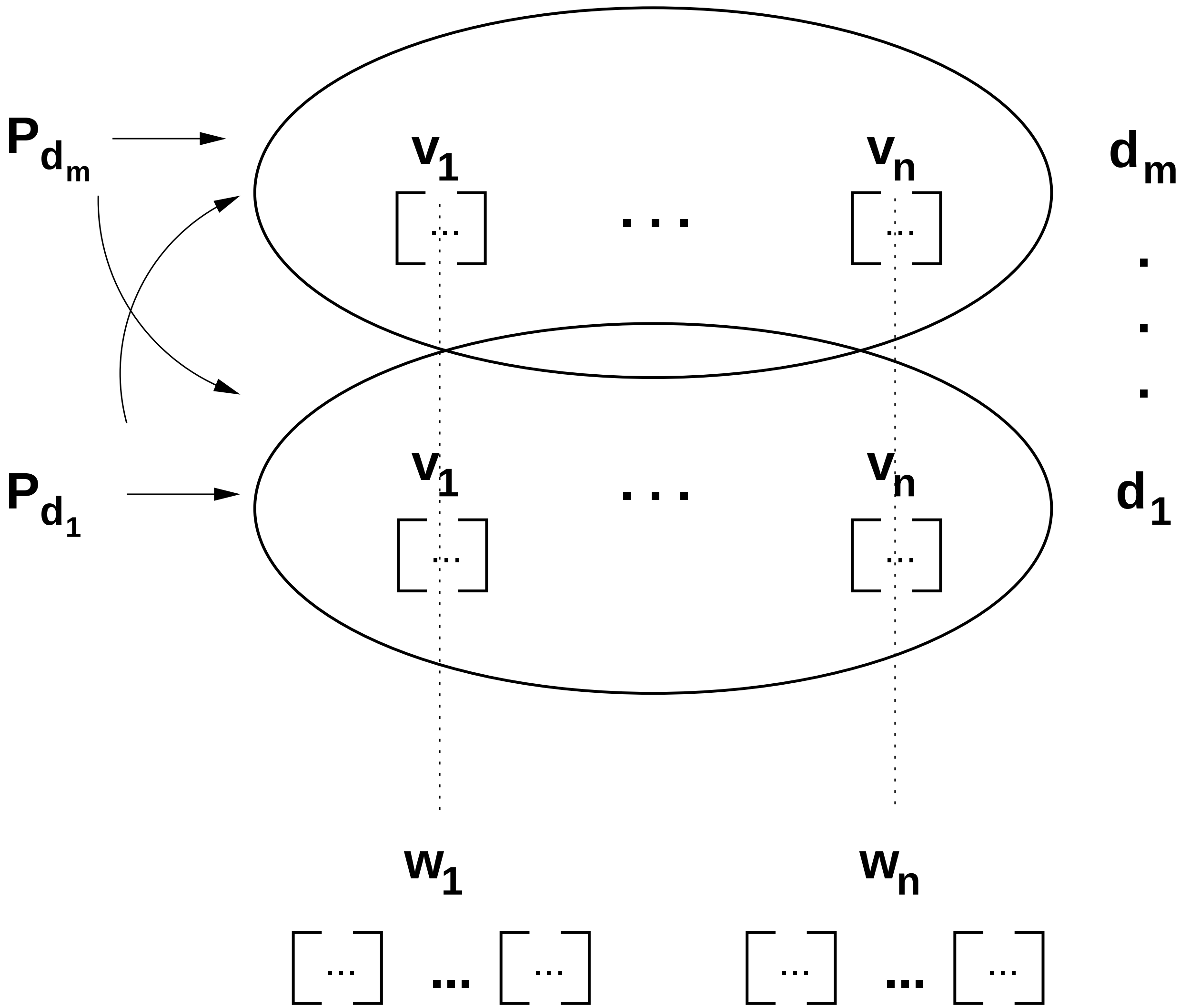
XDG architecture so far



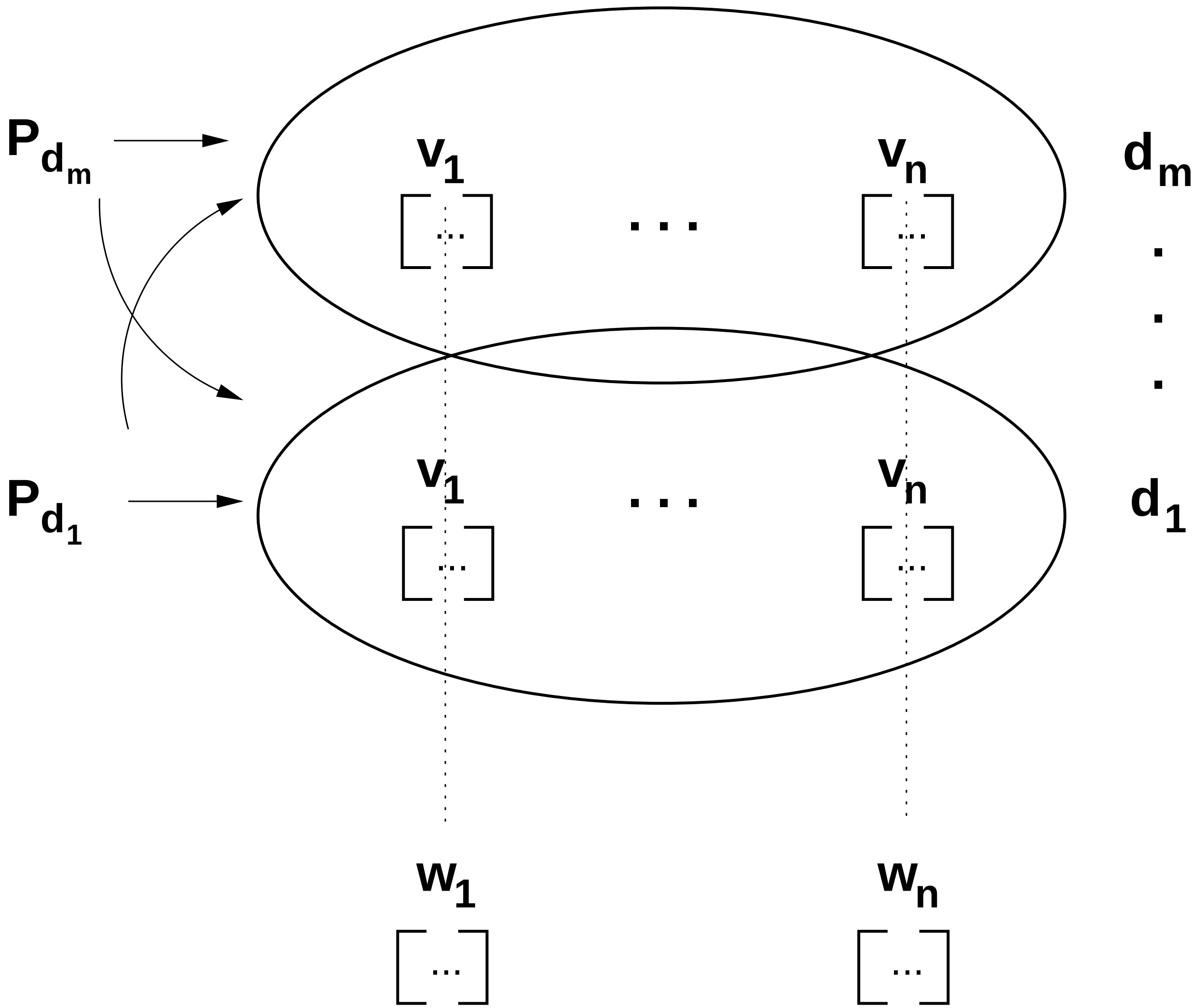
Words



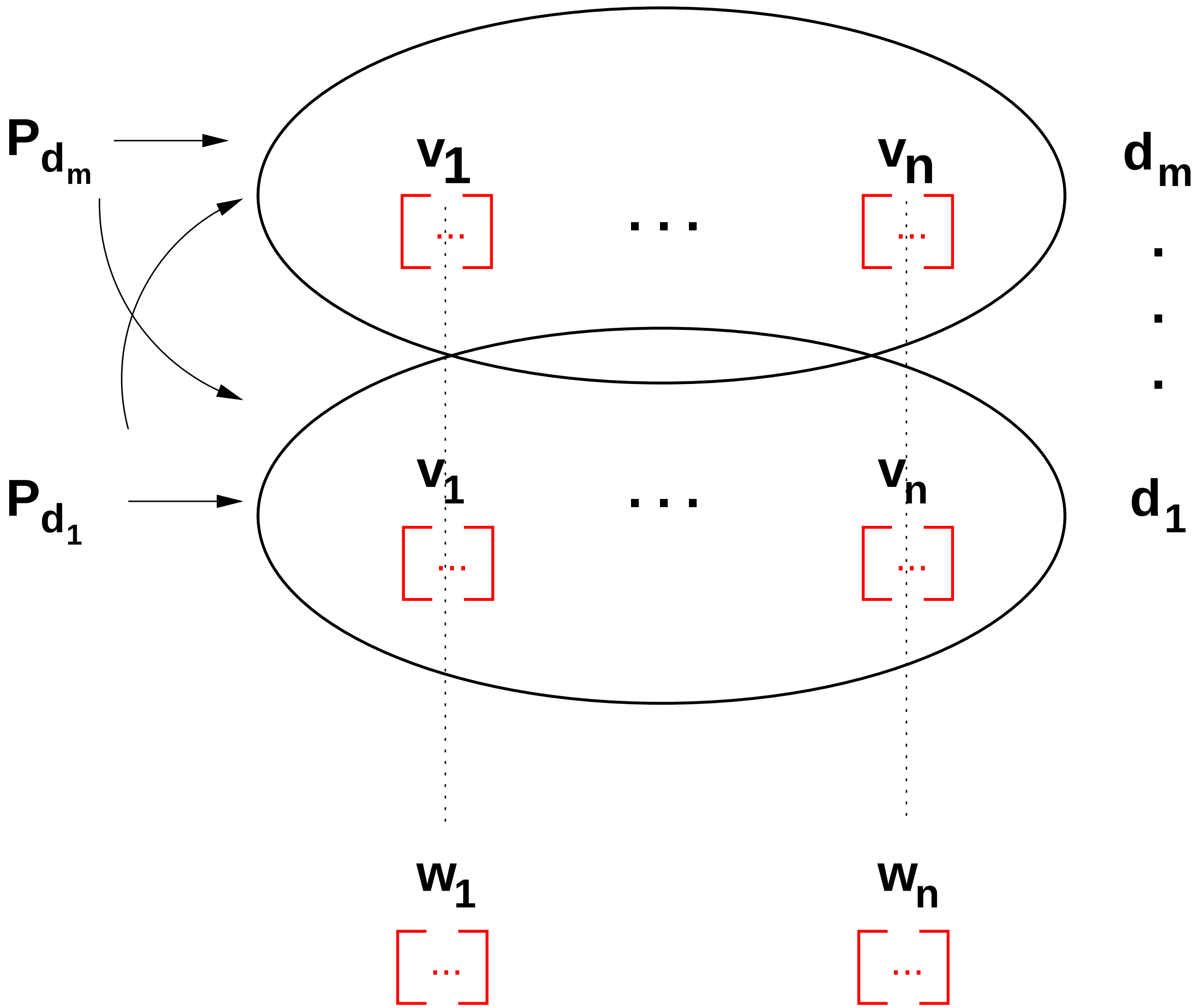
Lexical entries



Selection



Lexical assignment



XDG instantiation

- recipe for getting XDG instances:
  1. define graph dimensions
  2. define used principles and parameters

XDG does TDG

- two graph dimensions: G_{ID} and G_{LP}
- ID dimension: Immediate Dominance; edge labels: grammatical functions like subj, obj (subject, object)
- LP dimension: Linear Precedence; edge labels: topological fields (linear positions) like topf, subjf (topicalisation field, subject field)

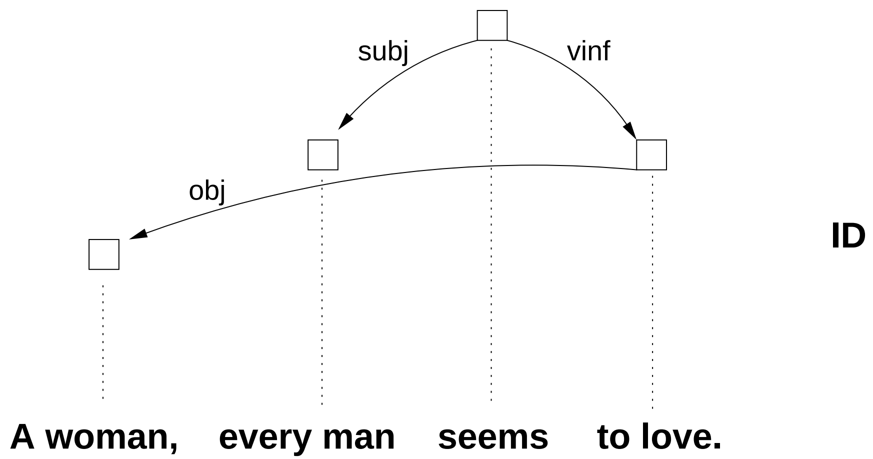
Principles used on the ID dimension

- $\mathsf{tree}(G_{\mathsf{ID}})$
- $\mathsf{in}(G_{\mathsf{ID}}, \mathsf{in}_{\mathsf{ID}})$
- $\mathsf{out}(G_{\mathsf{ID}}, \mathsf{out}_{\mathsf{ID}})$
- $\mathsf{nodeconstraints}(\ldots)$
- $\mathsf{edgeconstraints}(G_{\mathsf{ID}}, \mathsf{f})$

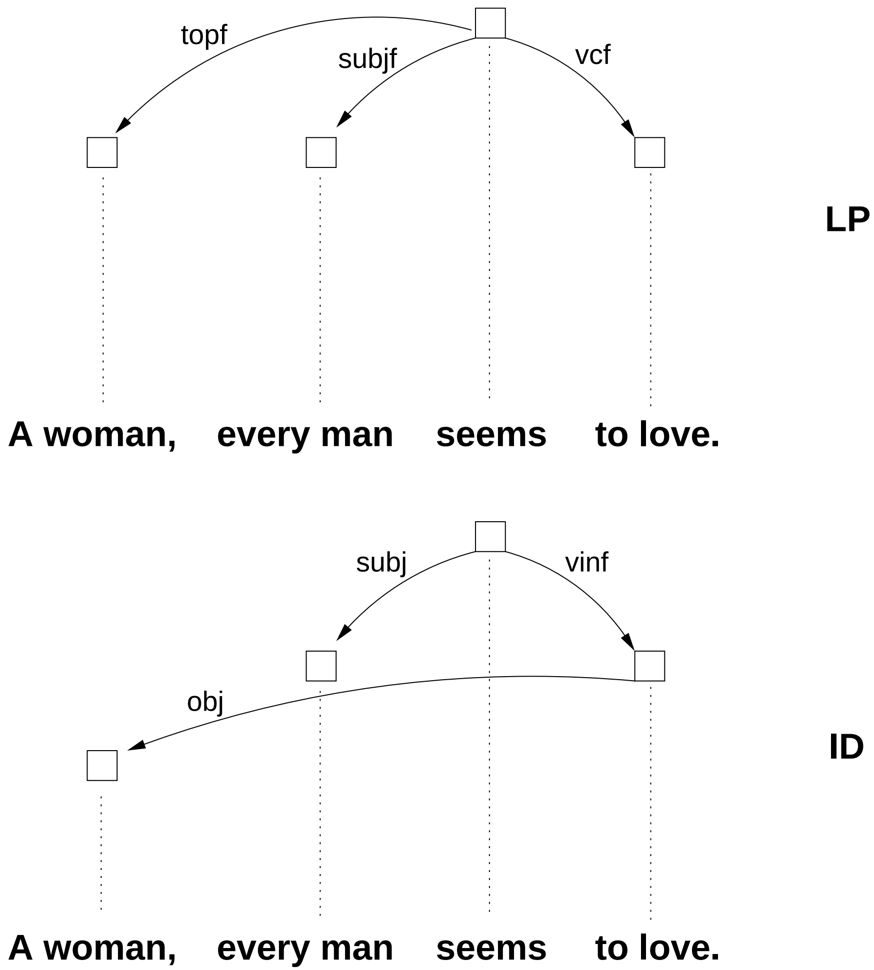
Principles used on the LP dimension

- $\mathsf{tree}(G_{\mathsf{LP}})$
- $\mathsf{in}(G_{\mathsf{LP}}, \mathsf{in}_{\mathsf{LP}})$
- $\mathsf{out}(G_{\mathsf{LP}}, \mathsf{out}_{\mathsf{LP}})$
- $\mathsf{order}(G_{\mathsf{LP}}, \ldots, \mathsf{on})$
- $\mathsf{projectivity}(G_{\mathsf{LP}})$
- $\mathsf{climbing}(G_{\mathsf{ID}}, G_{\mathsf{LP}})$
- $\mathsf{barriers}(G_{\mathsf{ID}}, G_{\mathsf{LP}}, \mathsf{blocks})$

TDG analysis



TDG analysis



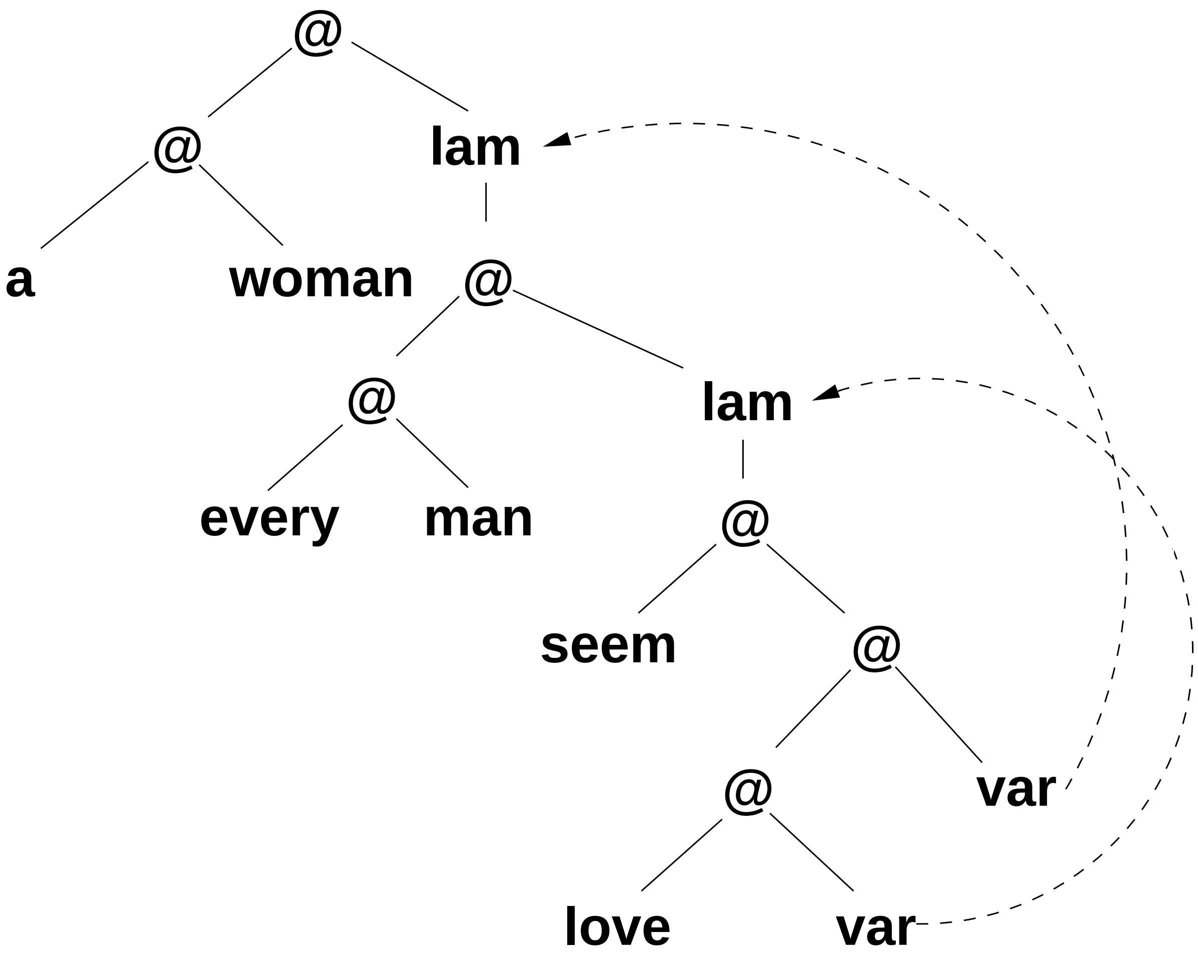
Syntax-semantics interface

- Semantic Topological Dependency Grammar (STDG)
- new grammar formalism, extends TDG with a syntax-semantics interface to underspecified semantics
- underspecification formalism: Constraint Language for Lambda Structures (CLLS, Egg et al 1998)
- other target semantics formalisms possible

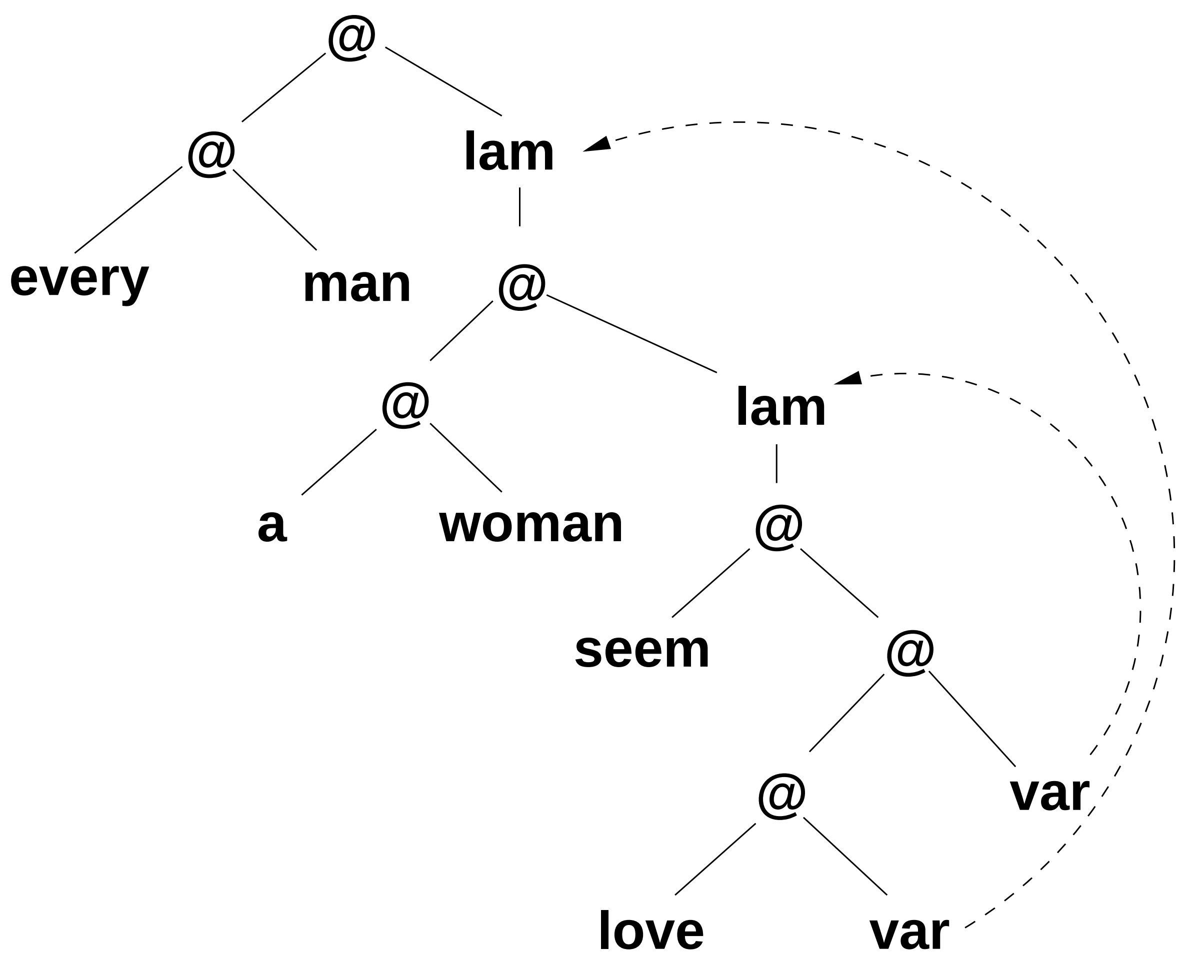
Constraint Language for Lambda Structures (CLLS)

- CLLS based on dominance constraints (Marcus/Hindle/Fleck 1983)
- CLLS structures describe λ-terms
- example: *A woman, every man seems to love.*
- scopally ambiguous: strong and weak reading (quantifier order: $\exists\forall$ and $\forall\exists$)

Strong reading



Weak reading



XDG does STDG

- four graph dimensions: G_{ID}, G_{LP}, G_{TH}, G_{DE}
- ID and LP dimensions as in TDG
- TH dimension: THematic dag; edge labels: semantic roles like act, pat (actor, patient)
- DE dimension: CLLS DErivation tree; edge labels: CLLS fragment positions like r, s (restriction, scope)

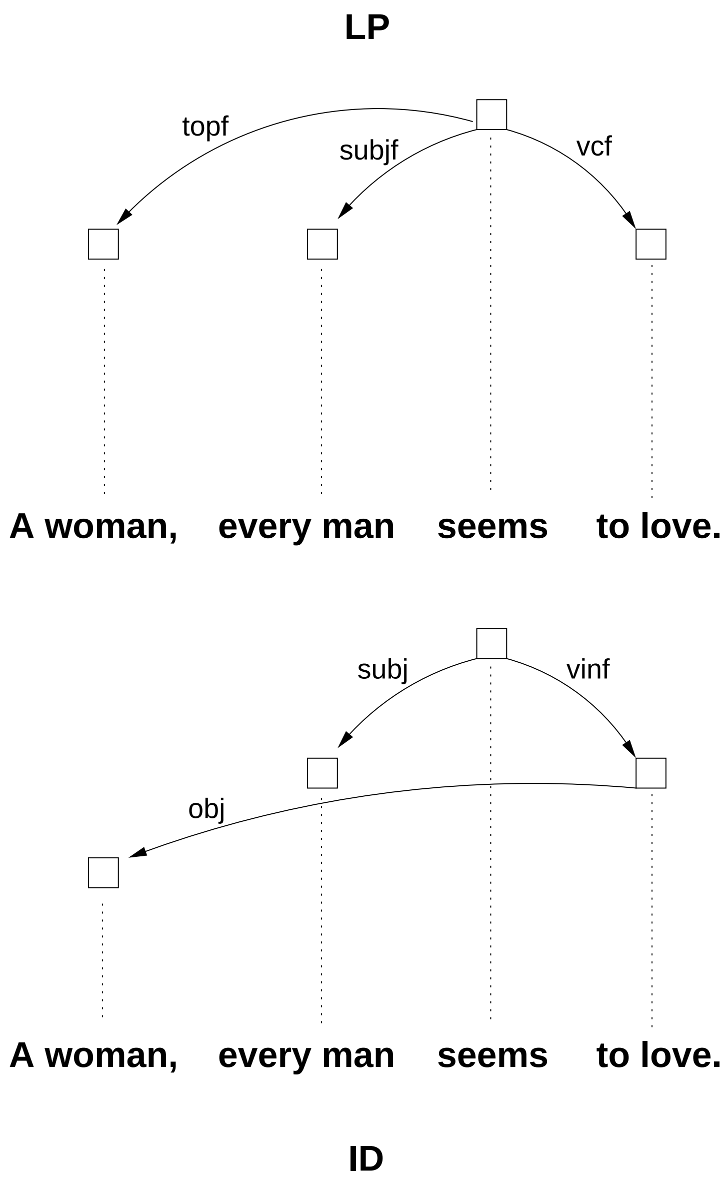
Principles used on the TH dimension

- $\mathsf{dag}(G_{\mathsf{TH}})$
- $\mathsf{in}(G_{\mathsf{TH}}, \mathsf{in}_{\mathsf{TH}})$
- $\mathsf{out}(G_{\mathsf{TH}}, \mathsf{out}_{\mathsf{TH}})$
- $\mathsf{linking}(G_{\mathsf{TH}}, G_{\mathsf{ID}}, \mathsf{real}, \mathsf{subs})$

Principles used on the DE dimension

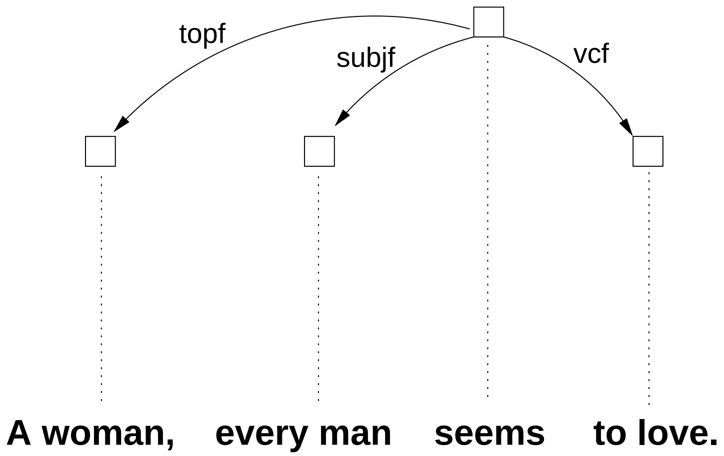
- $\mathsf{tree}(G_{\mathsf{DE}})$
- $\mathsf{in}(G_{\mathsf{DE}}, \mathsf{in}_{\mathsf{DE}})$
- $\mathsf{out}(G_{\mathsf{DE}}, \mathsf{out}_{\mathsf{DE}})$
- $\mathsf{covariance}(G_{\mathsf{DE}}, G_{\mathsf{ID}}, \mathsf{co})$
- $\mathsf{contravariance}(G_{\mathsf{DE}}, G_{\mathsf{ID}}, \mathsf{contra})$

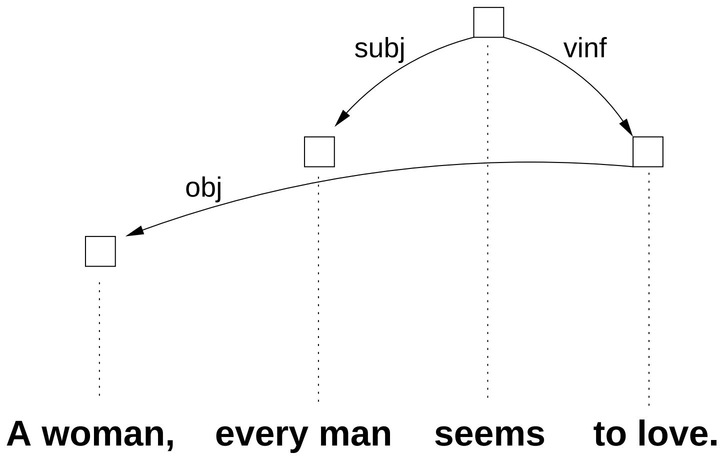
STDG analysis



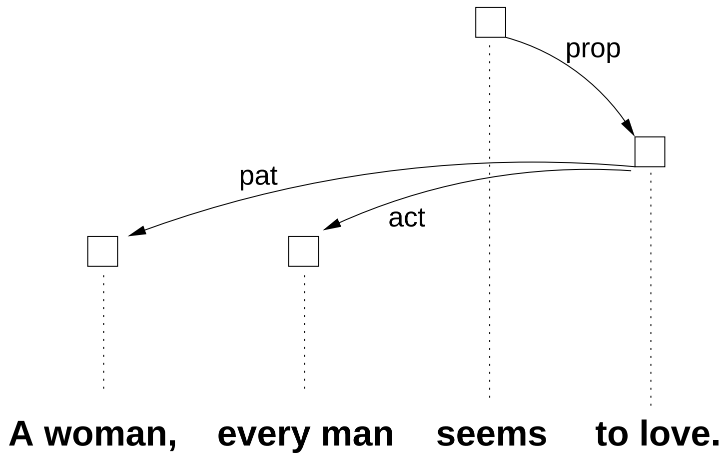
STDG analysis

LP



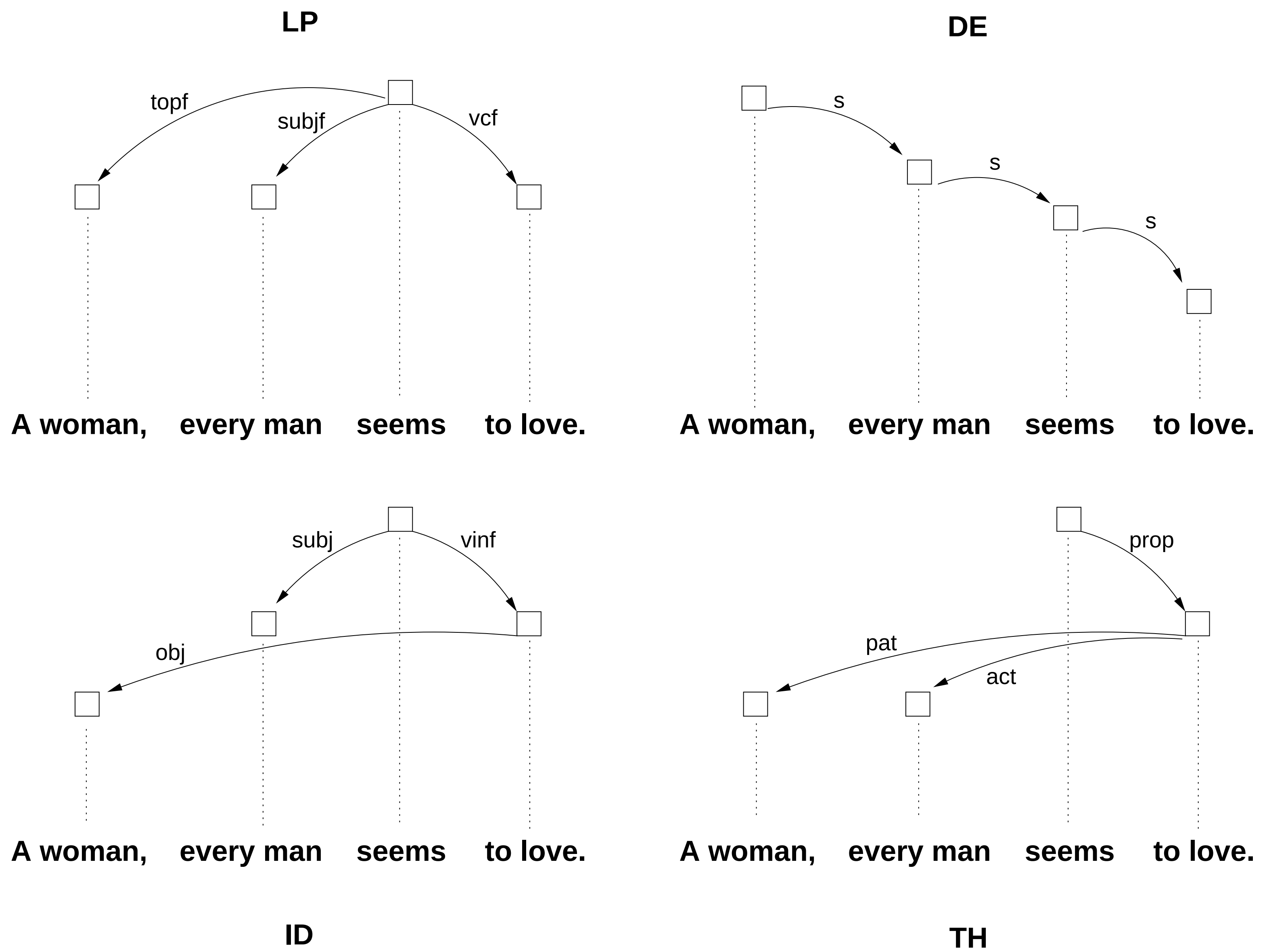


ID

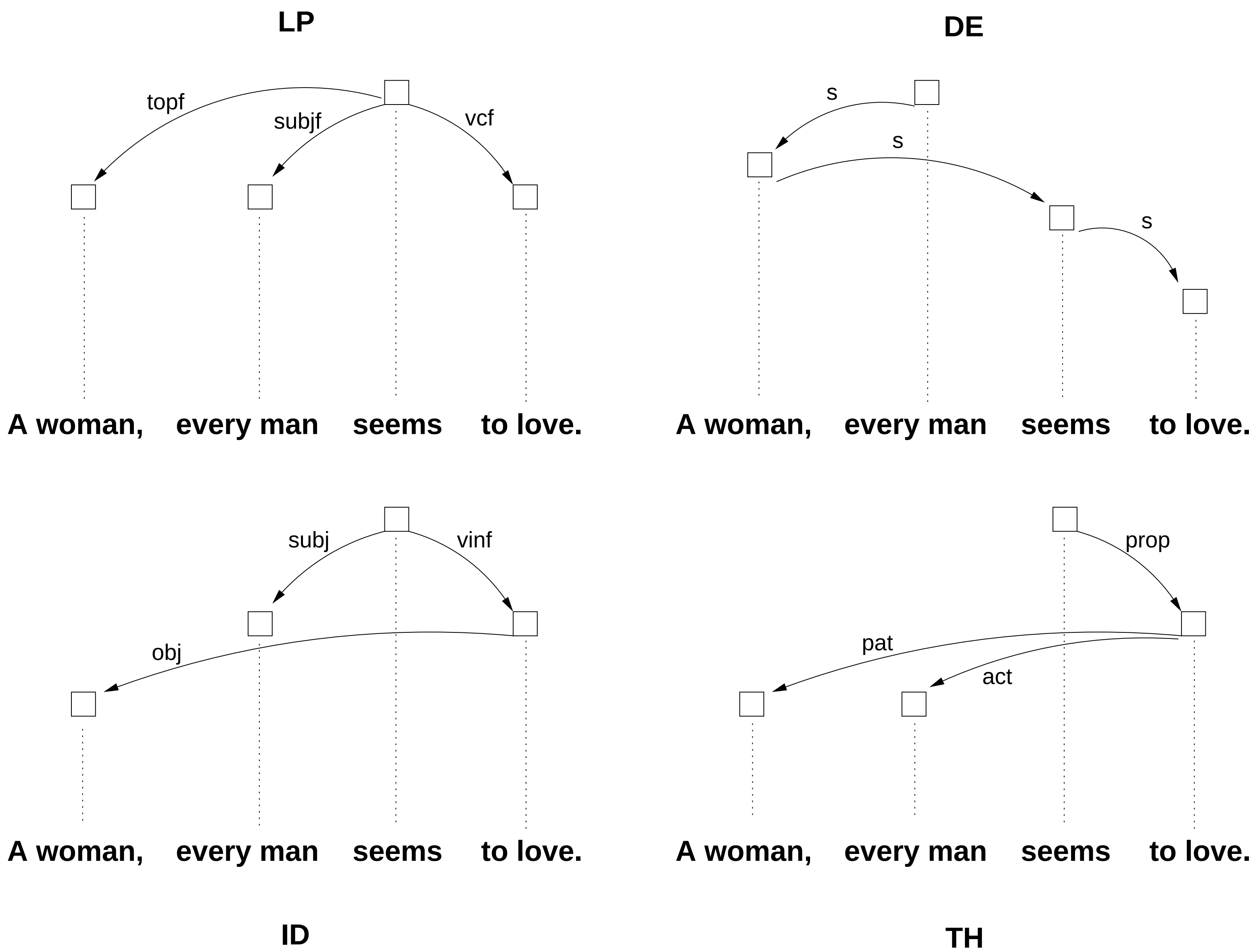


TH

STDG analysis (strong reading)



STDG analysis (weak reading)

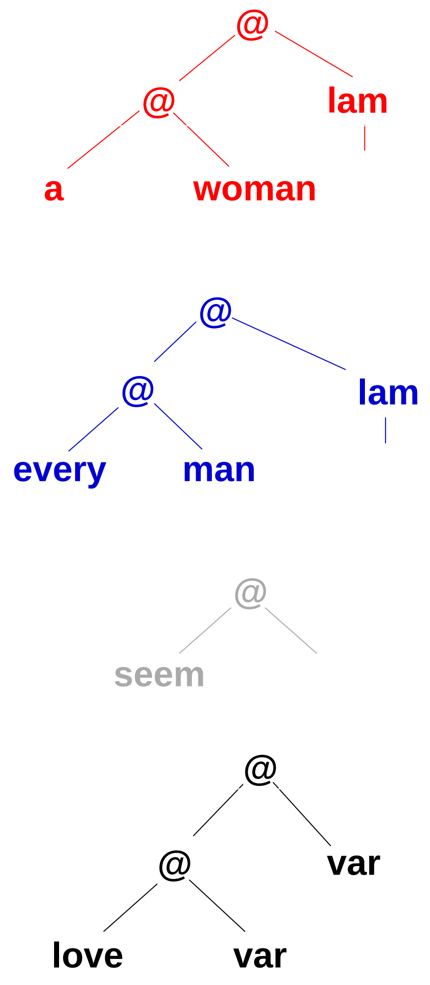


From STDG to CLLS

- lexicon: words correspond to CLLS fragments (subtrees)
- STDG analysis contains all information to build a CLLS representation of the semantics:
    - DE tree: assembly of fragments/scope
    - TH dag: lambda bindings

Words correspond to CLLS fragments

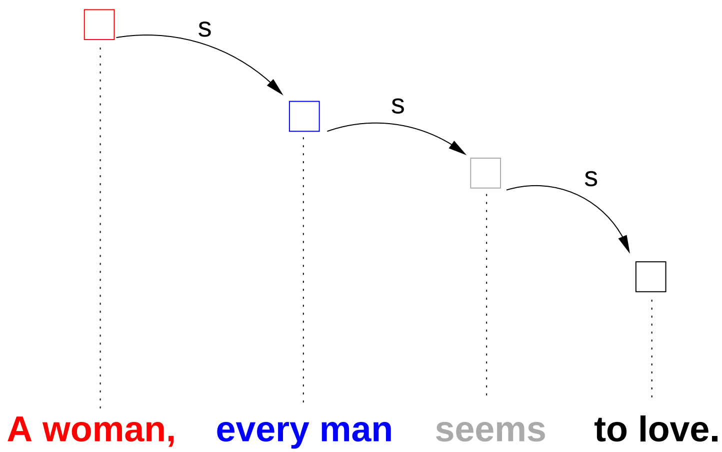
A woman, every man seems to love.

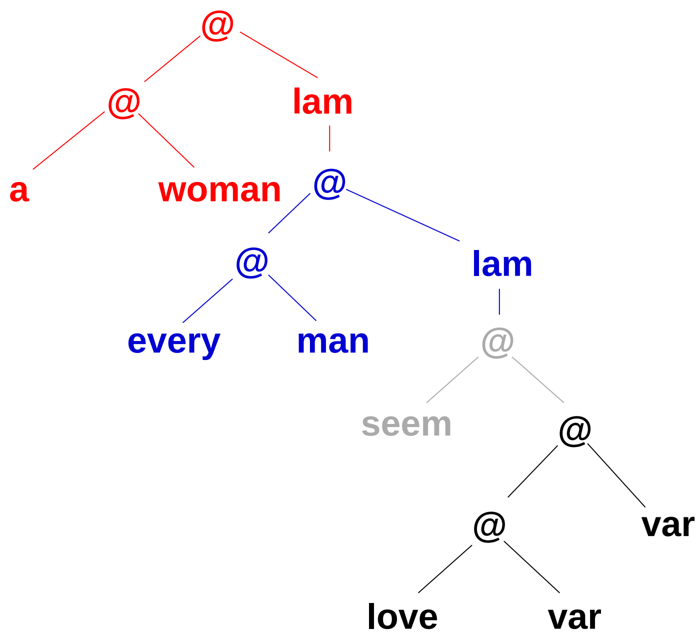


CLLS

DE tree: assembly of fragments

DE

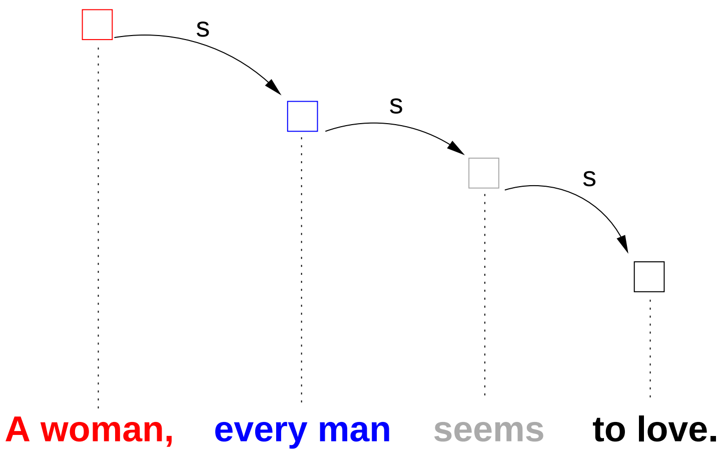


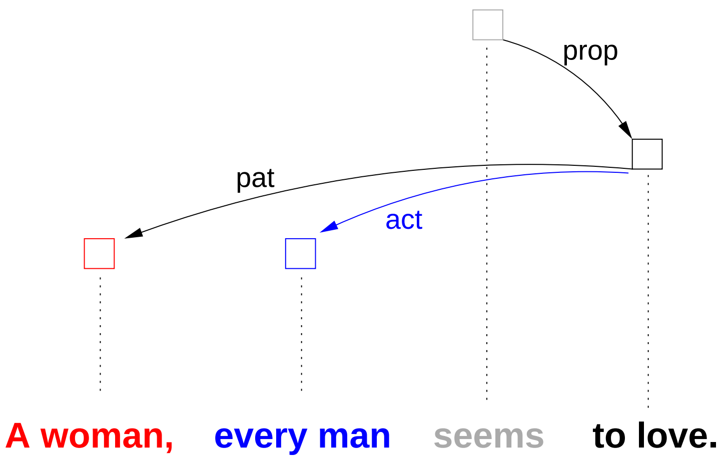


CLLS

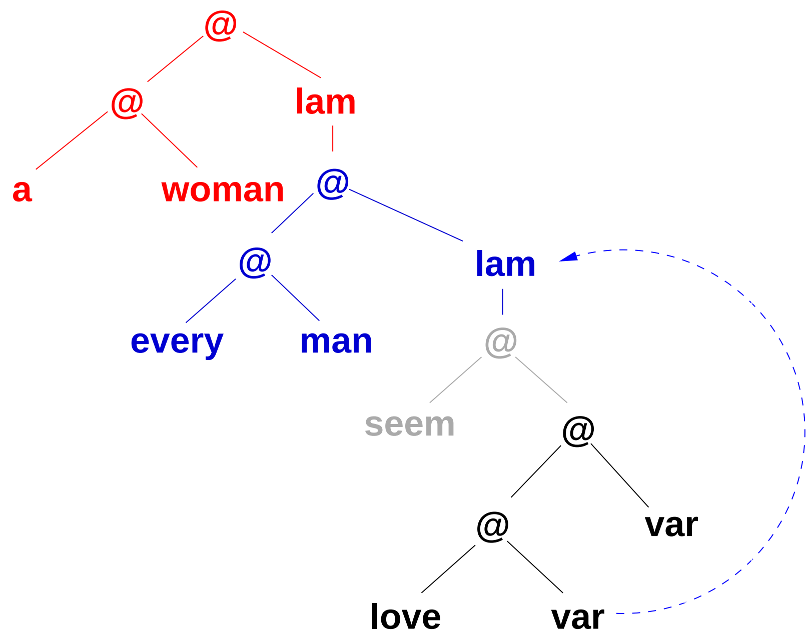
TH dag: lambda bindings

DE





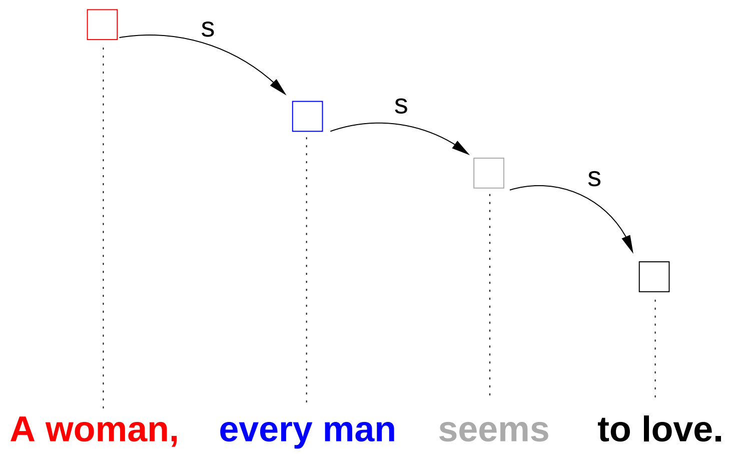
TH

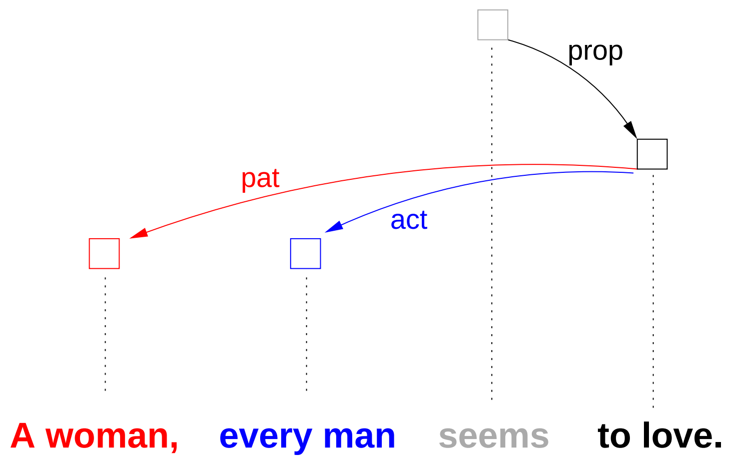


CLLS

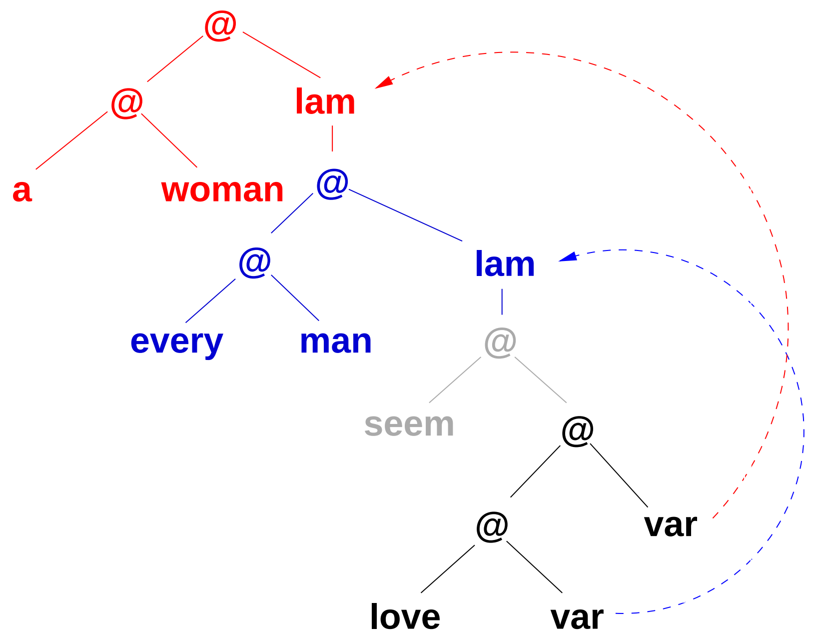
TH dag: lambda bindings

DE





TH



CLLS

Summary

- dependency grammar appealing but pure dependency grammar approaches flawed
- TDG solves the word order problem, but still no syntax-semantics interface
- generalised TDG to XDG
- TDG as an instance of XDG
- syntax-semantics interface: developed STDG as another instance of XDG

State of the art

- proof of concept: STDG syntax-semantics interface works for small example grammar
- new XDG parser system (as efficient as the TDG parser)
- demo offline for those interested

Related work

- interface to information structure (Duchier and Kruijff 2003)
- grammar induction (Korthals 2003)
- Stochastic eXtensible Dependency Grammar (SXDG) (Dienes, Koller and Kuhlmann 2003)

Outlook

- integration of preferences (for e.g. PP attachment, scope)
- search for equivalences between instances of XDG and existing grammar formalisms (find e.g. context-free and mildly context-sensitive XDG instances)
  - Tree Insertion Grammar (TIG, Schabes and Waters 1993)
  - TAG
  - CCG (Steedman 2000), MMCCG (Baldridge and Kruijff 2003)
- development of bigger grammars:
  - handcrafted
  - induced
  - ported

Thank you!

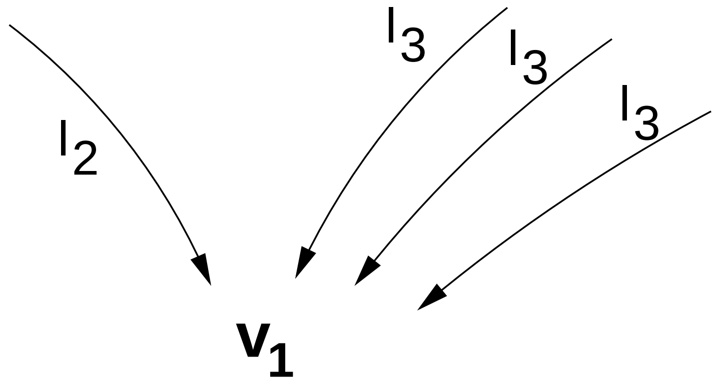
Any questions?

Extra slides

In

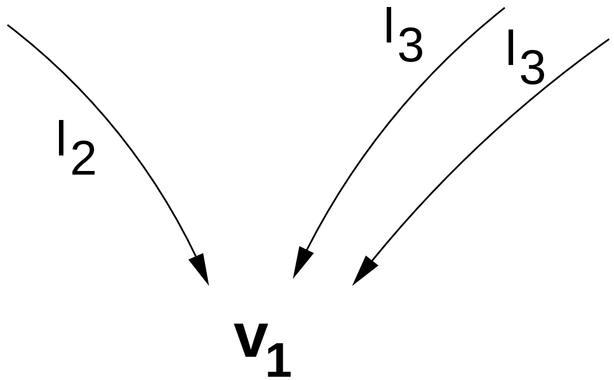
$\mathsf{out}(G_d, \mathsf{f})$: The incoming edges of each node in G_d must satisfy the nodes' *in specification*. Feature $\mathsf{f} : V \rightarrow (L_d \rightarrow 2^{\mathbb{N}})$ maps an in specification to each node.

In



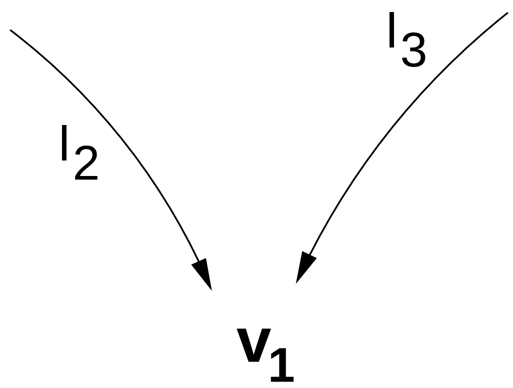
$$\left[\ \text{in} : \left[\begin{array}{l} l_1 : \{0\} \\ l_2 : \{1\} \\ l_3 : \{0,1,2,3\} \end{array}\right]\ \right]$$

In



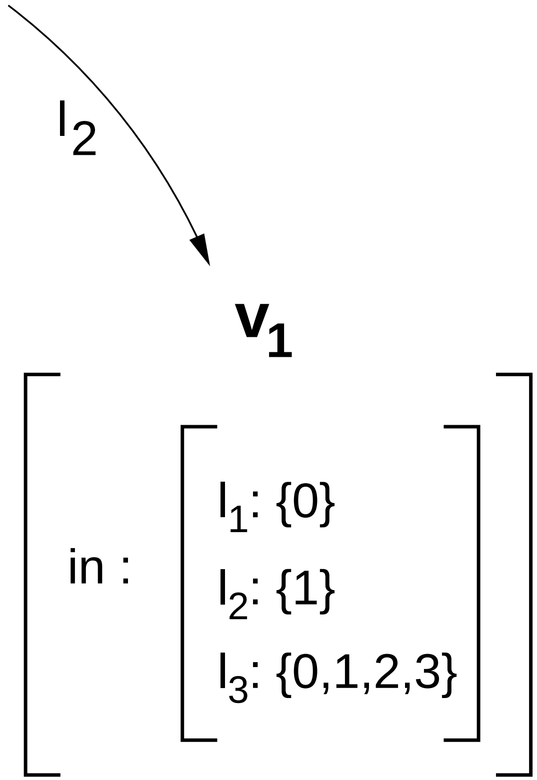
$$\left[\text{in} : \begin{bmatrix} l_1 : \{0\} \\ l_2 : \{1\} \\ l_3 : \{0,1,2,3\} \end{bmatrix} \right]$$

In

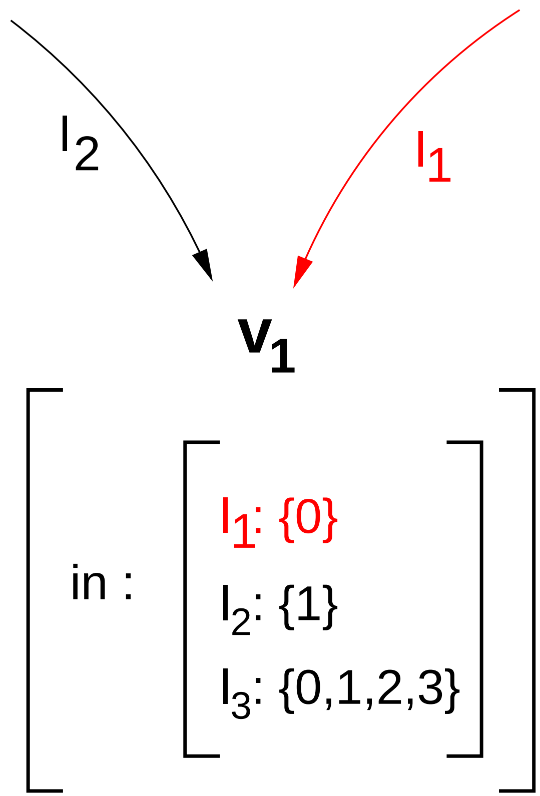


$$\left[\ \text{in}: \begin{bmatrix} l_1 : \{0\} \\ l_2 : \{1\} \\ l_3 : \{0,1,2,3\} \end{bmatrix}\ \right]$$

In



In



In

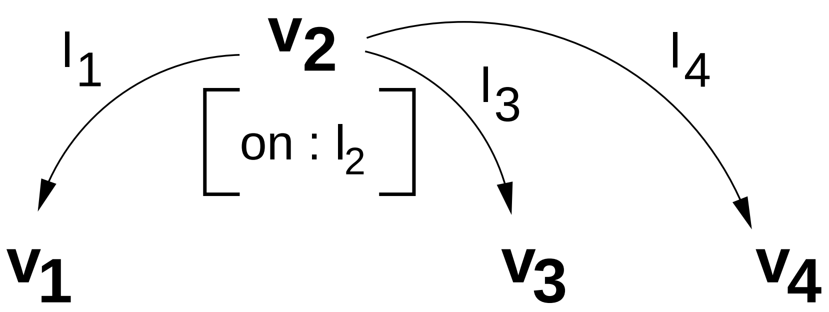
$$\mathbf{v_1}\ \left[\ \text{in}: \begin{bmatrix} l_1: \{0\} \\ l_2: \{1\} \\ l_3: \{0,1,2,3\} \end{bmatrix} \right]$$

Order

$\mathsf{order}(G_d, \prec, \mathsf{f})$: The daughters of each node v in G_d must be ordered according to their edge label, and v itself according to its node label, and the total order L_d stipulated in $\prec$. Feature $\mathsf{f} : V \to L_d$ assigns a node label to each node. We call f the *on specification* of a node.

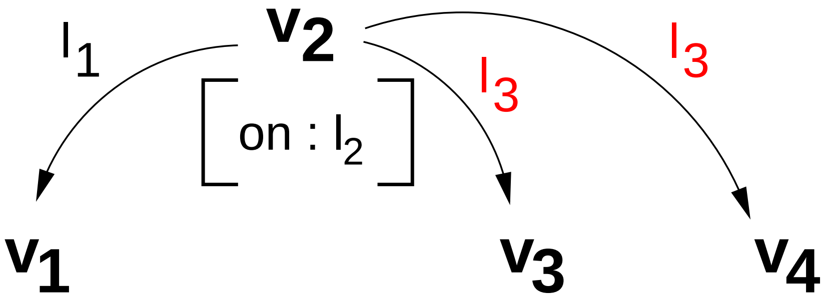
Order

$$l_1 < l_2 < l_3 < l_4$$



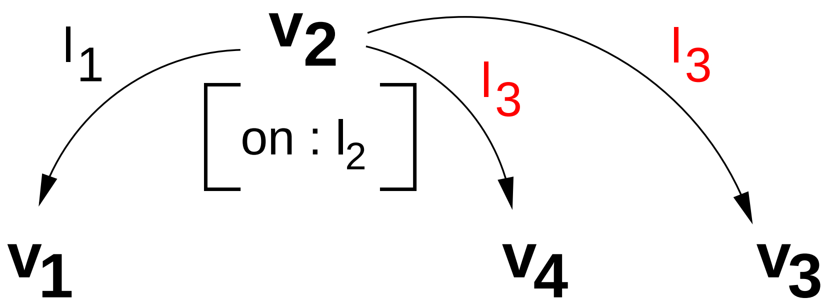
Order

$$l_1 < l_2 < l_3 < l_4$$



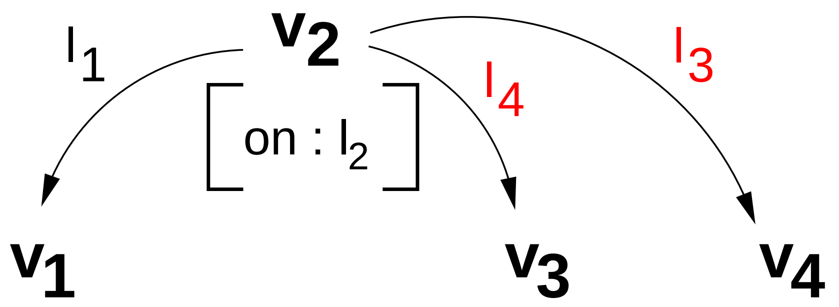
Order

$$l_1 < l_2 < l_3 < l_4$$



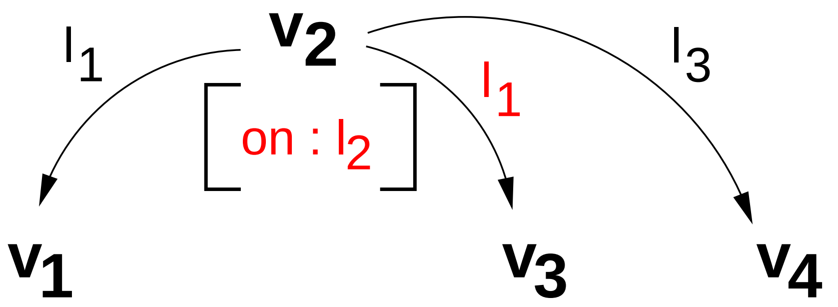
Order

$$l_1 < l_2 < l_3 < l_4$$



Order

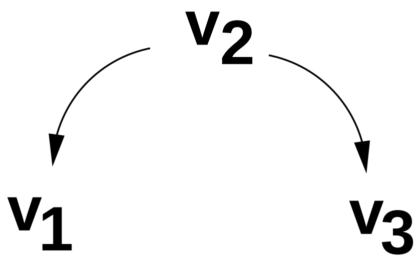
$l_1 < l_2 < l_3 < l_4$



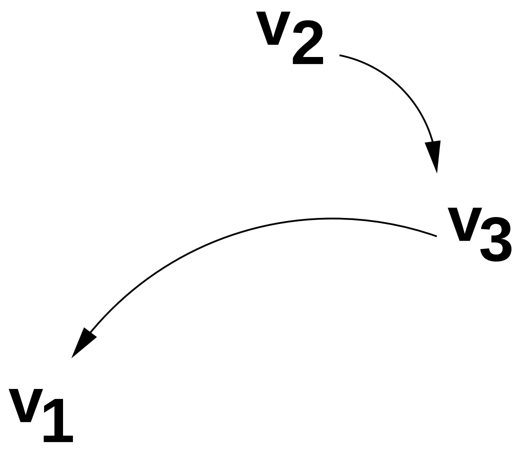
Projectivity

$\mathsf{projectivity}(G)$: G must be projective.

Projectivity



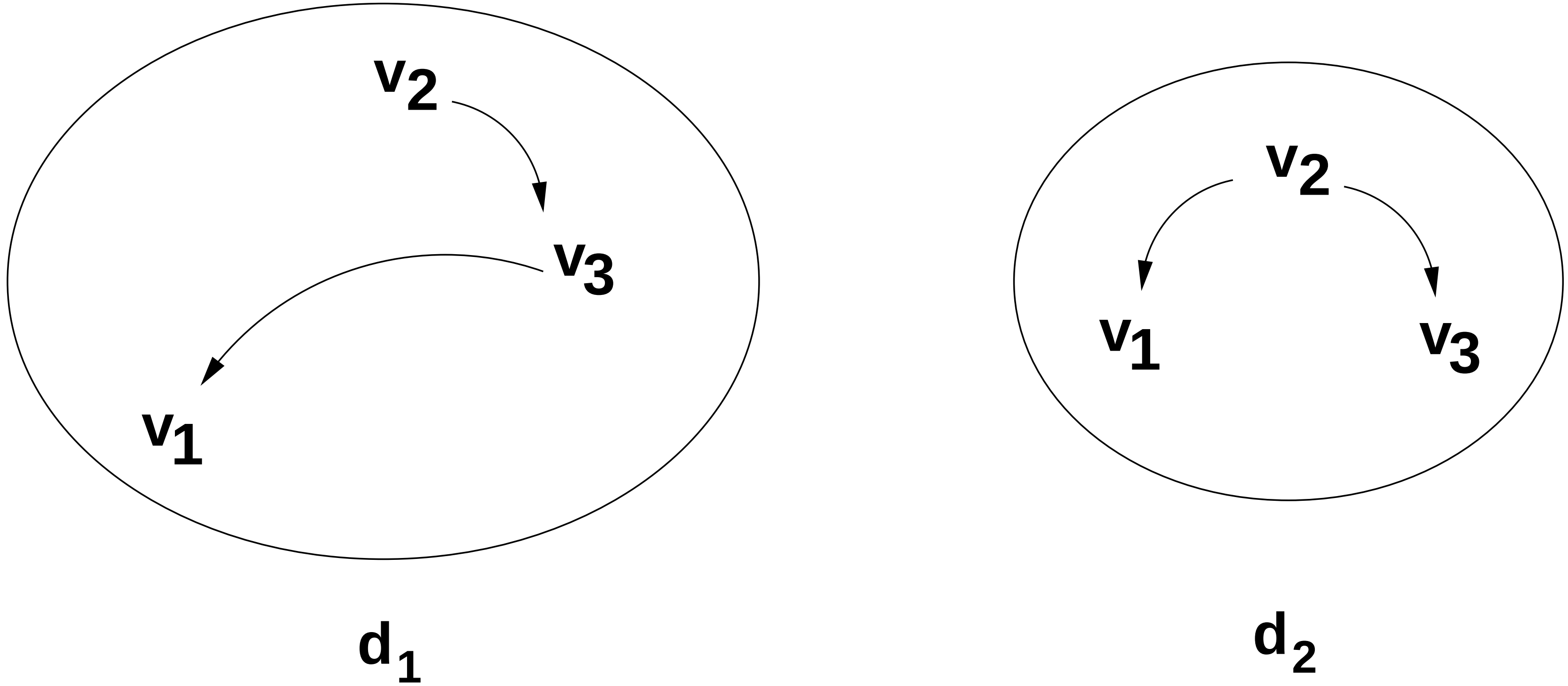
Projectivity



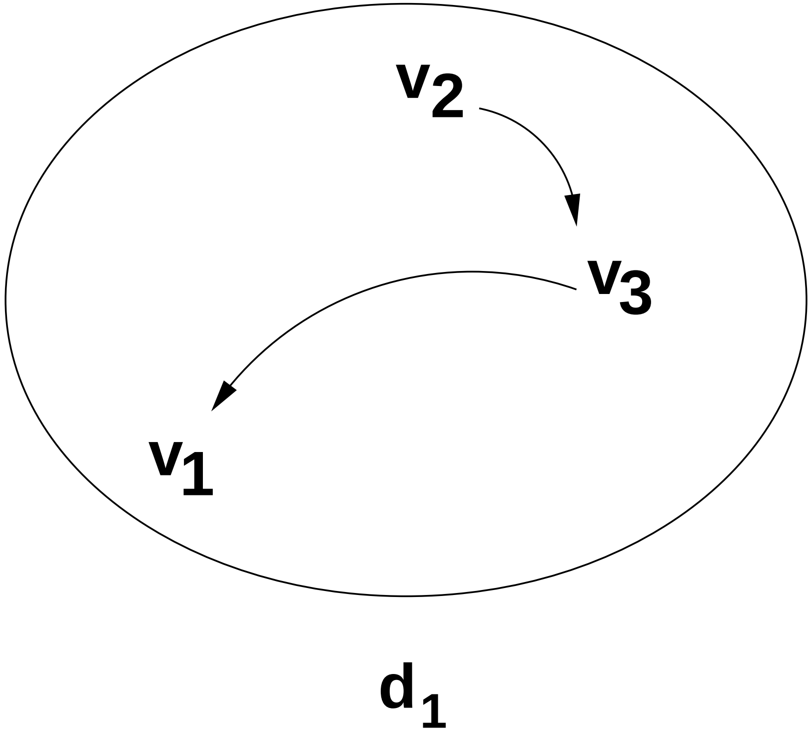
Climbing

climbing(G_{d_1}, G_{d_2}): G_{d_2} must be flatter than G_{d_1}.

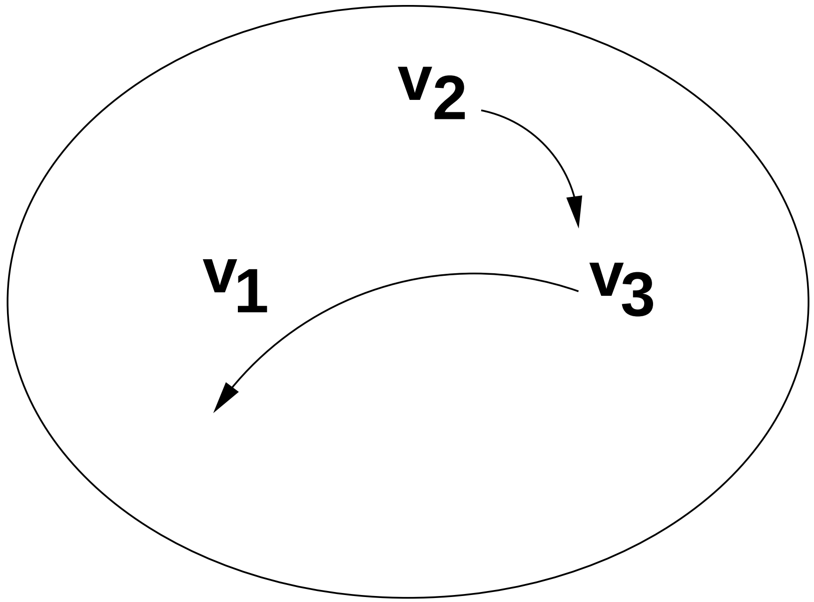
Climbing



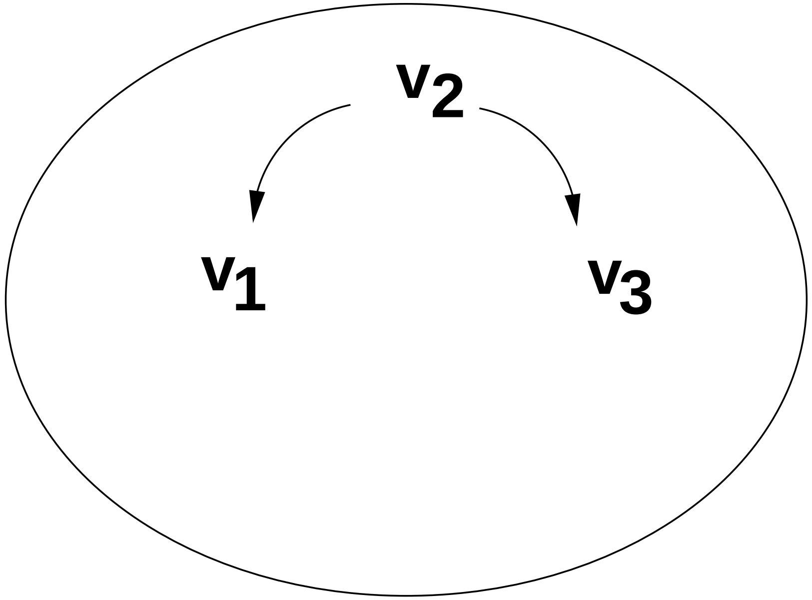
Climbing



Climbing



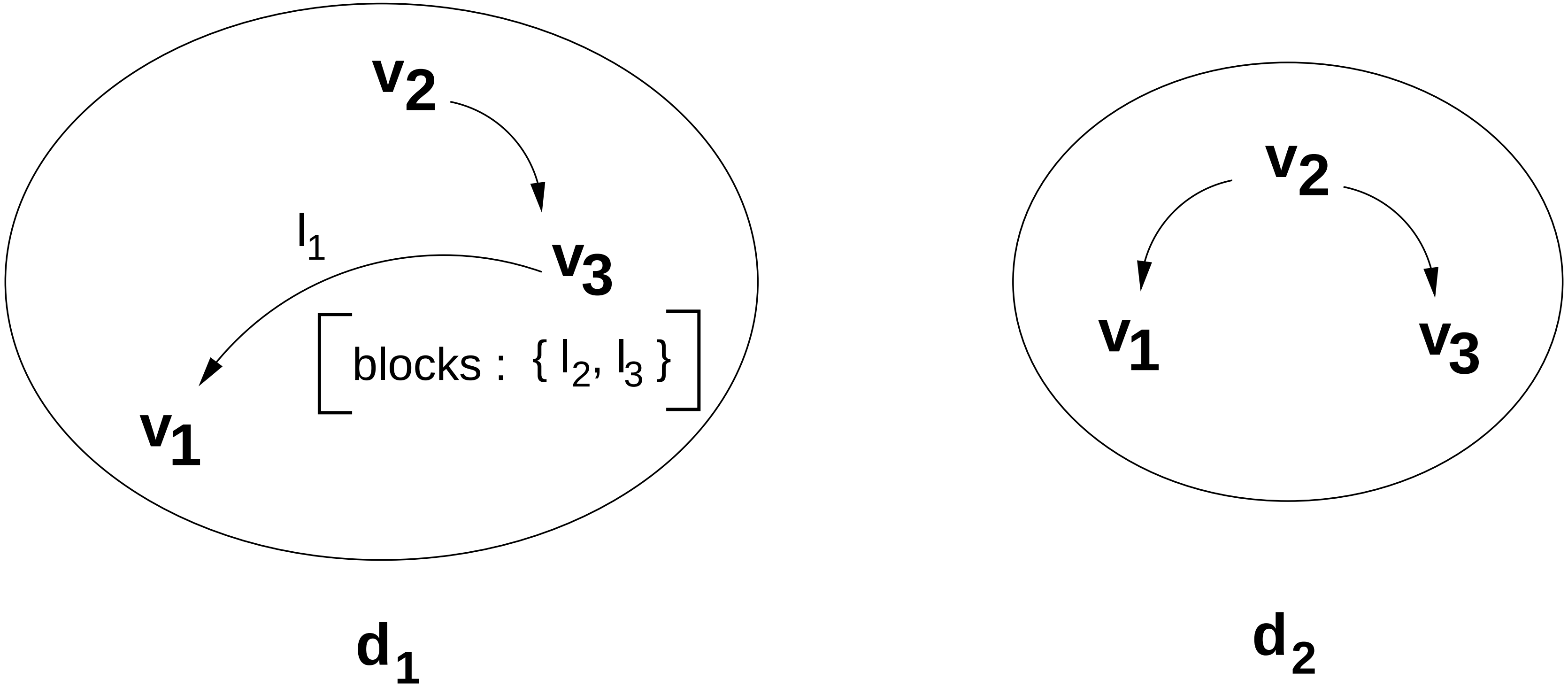
Climbing



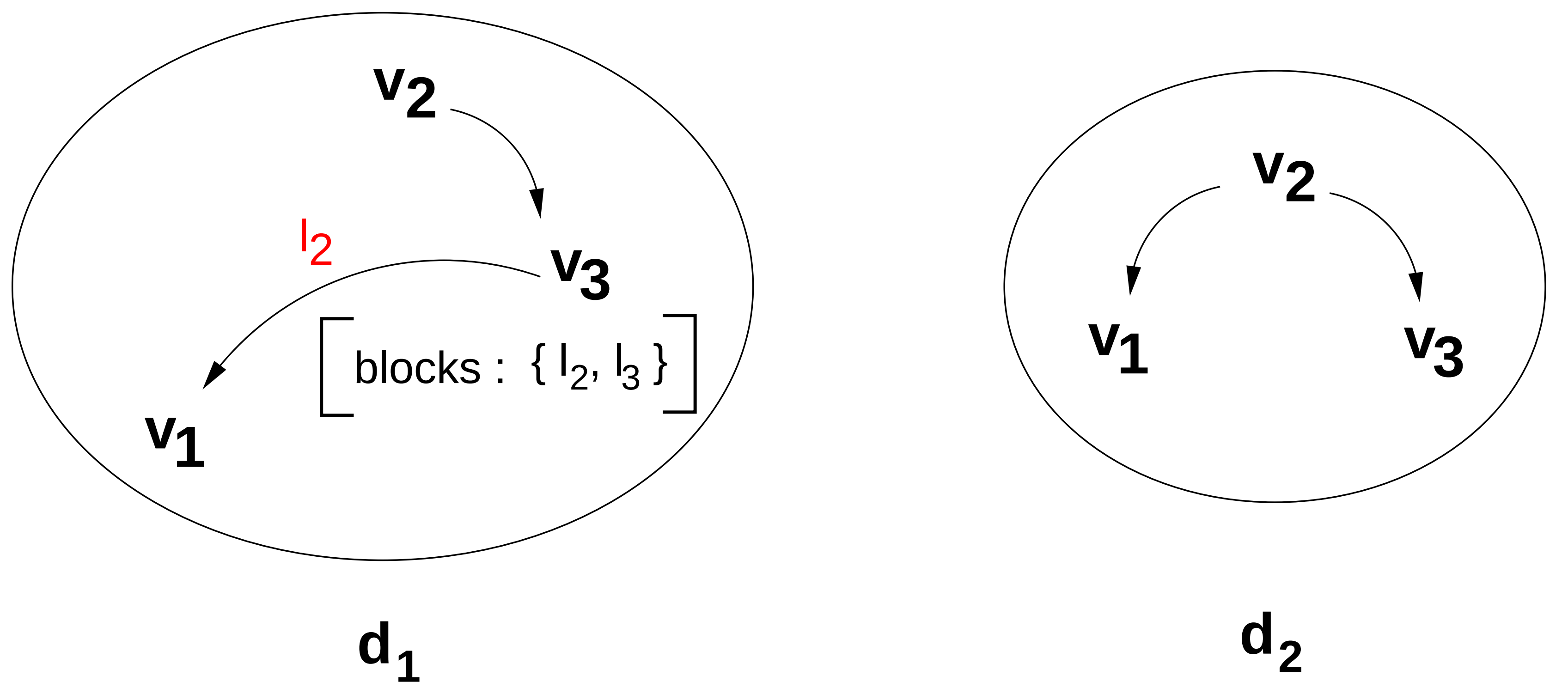
Barriers

$\mathsf{barriers}(G_{d_1}, G_{d_2}, \mathsf{f})$: No node may climb through a barrier. Feature $\mathsf{f} : V \rightarrow 2^{L_{d_1}}$ assigns to each node the set of labels for which it acts as a barrier. We call f the *blocking specification* of a node.

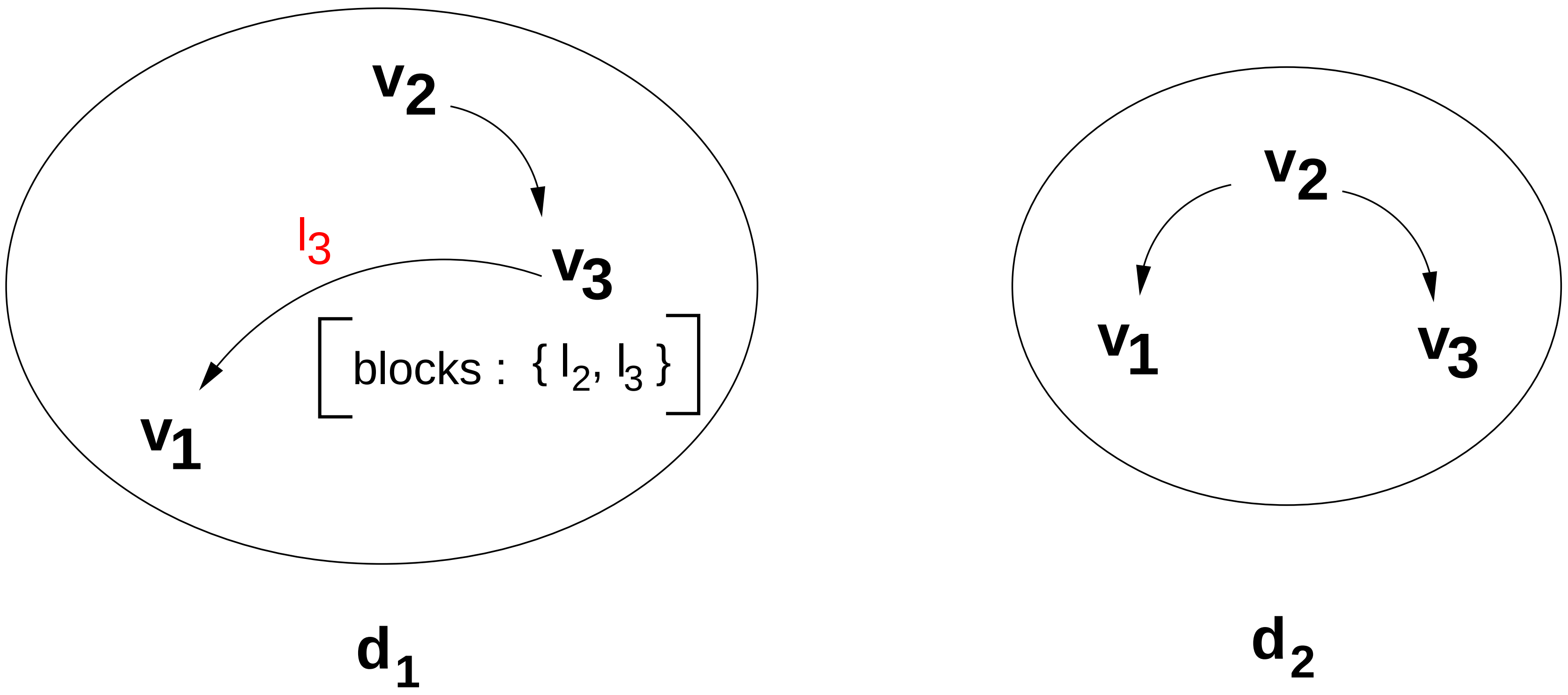
Barriers



Barriers



Barriers



Node constraints

$\mathsf{nodeconstraints}(2^c)$: Each node must satisfy a set of node constraints written in the simple constraint language C:

$$C \quad ::= \quad \begin{array}{ccc} x & = & y \\ x & \neq & y \\ x & \in & y \\ x & \notin & y \\ x & \subseteq & y \\ x & \parallel & y \end{array}$$

Edge constraints

$\mathsf{edgeconstraints}(G_d, \mathsf{f})$: Each edge (v_1, l, v_2) in G_d must satisfy a set of edge constraints written in constraint language C. Function $\mathsf{f} : L_d \rightarrow 2^C$ maps edge labels to sets of constraints.