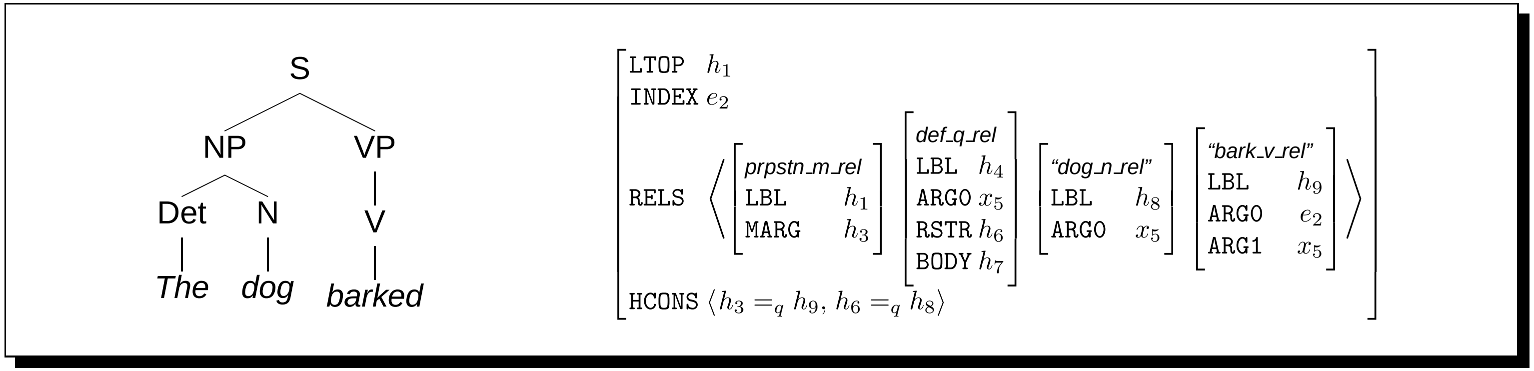


Computational Linguistics (INF2820 — TFSs)

$$\textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\rangle\\ \text{COMPS} & \boxed{3}\end{bmatrix} \longrightarrow \boxed{2}\,\textit{phrase}\begin{bmatrix}\text{SPR} & \langle\rangle\\ \text{COMPS} & \langle\rangle\end{bmatrix},\ \textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\boxed{2}\rangle\\ \text{COMPS} & \boxed{3}\end{bmatrix}$$

Stephan Oepen

Universitetet i Oslo

oe@ifi.uio.no

Feature Structure Subsumption: Examples

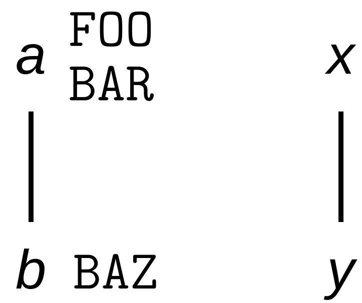
TFS$_1$: $a\begin{bmatrix} \text{FOO } x \\ \text{BAR } x \end{bmatrix}$

TFS$_2$: $a\begin{bmatrix} \text{FOO } x \\ \text{BAR } y \end{bmatrix}$

TFS$_3$: $b\begin{bmatrix} \text{FOO } y \\ \text{BAR } x \\ \text{BAZ } x \end{bmatrix}$

TFS$_4$: $a\begin{bmatrix} \text{FOO } \boxed{1}\, x \\ \text{BAR } \boxed{1} \end{bmatrix}$

Hierarchy



Feature structure F subsumes feature structure G ($F \sqsubseteq G$) iff: (1) if path p is defined in F then p is also defined in G and the type of the value of p in F is a supertype or equal to the type of the value of p in G, and (2) all paths that are reentrant in F are also reentrant in G.

Typed Feature Structure Unification

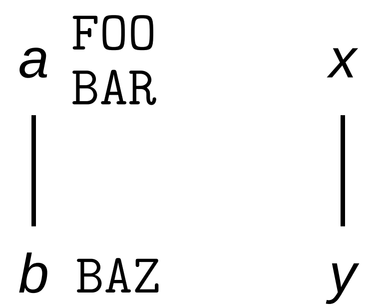
- Decide whether two typed feature structures are mutually compatible;
- determine combination of two TFSs to give the most general feature structure which retains all information which they individually contain;
- if there is no such feature structure, unification fails (depicted as $\bot$);
- unification *monotonically* combines information from both 'input' TFSs;
- *relation to subsumption* the unification of two structures *F* and *G* is the most general TFS which is subsumed by both *F* and *G* (if it exists).
- $\sqcap$ ('square set intersection') conventionally used to depict unification.

Typed Feature Structure Unification: Examples

TFS_1: $a\begin{bmatrix}\text{FOO } x \\ \text{BAR } x\end{bmatrix}$ TFS_2: $a\begin{bmatrix}\text{FOO } x \\ \text{BAR } y\end{bmatrix}$

TFS_3: $b\begin{bmatrix}\text{FOO } y \\ \text{BAR } x \\ \text{BAZ } x\end{bmatrix}$ TFS_4: $a\begin{bmatrix}\text{FOO } \boxed{1} x \\ \text{BAR } \boxed{1}\end{bmatrix}$

Hierarchy



$\text{TFS}_1 \sqcap \text{TFS}_2 \equiv \text{TFS}_2$ $\text{TFS}_1 \sqcap \text{TFS}_3 \equiv \text{TFS}_3$ $\text{TFS}_3 \sqcap \text{TFS}_4 \equiv b\begin{bmatrix}\text{FOO } \boxed{1} y \\ \text{BAR } \boxed{1} \\ \text{BAZ } x\end{bmatrix}$

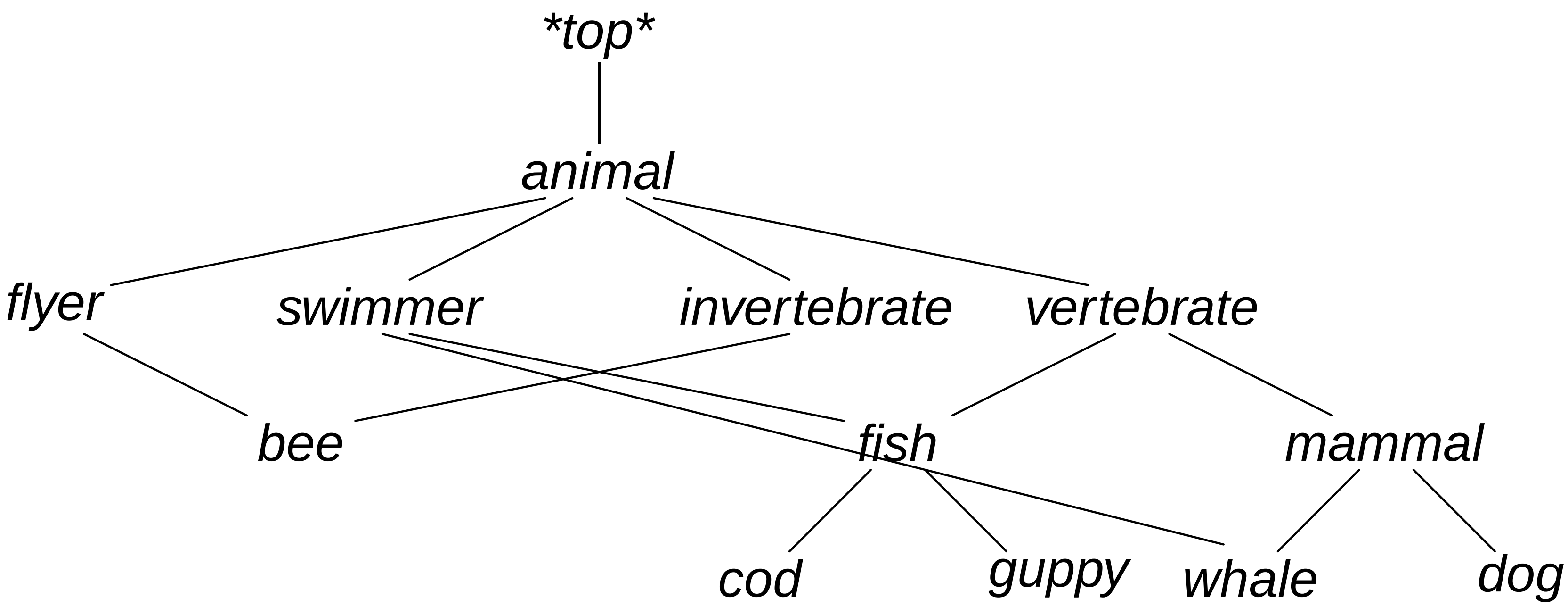
Type Constraints and Appropriate Features

- Well-formed TFSs satisfy all *type constraints* from the type hierarchy;
- type constraints are typed feature structures associated with a type;
- the top-level features of a type constraint are *appropriate features*;
- type constraints express generalizations over a ‘class’ (set) of objects.

type	**constraint**	**appropriate features**
ne-list	$\textit{*ne-list*}\begin{bmatrix}\texttt{FIRST} & \textit{*top*} \\ \texttt{REST} & \textit{*list*}\end{bmatrix}$	FIRST and REST

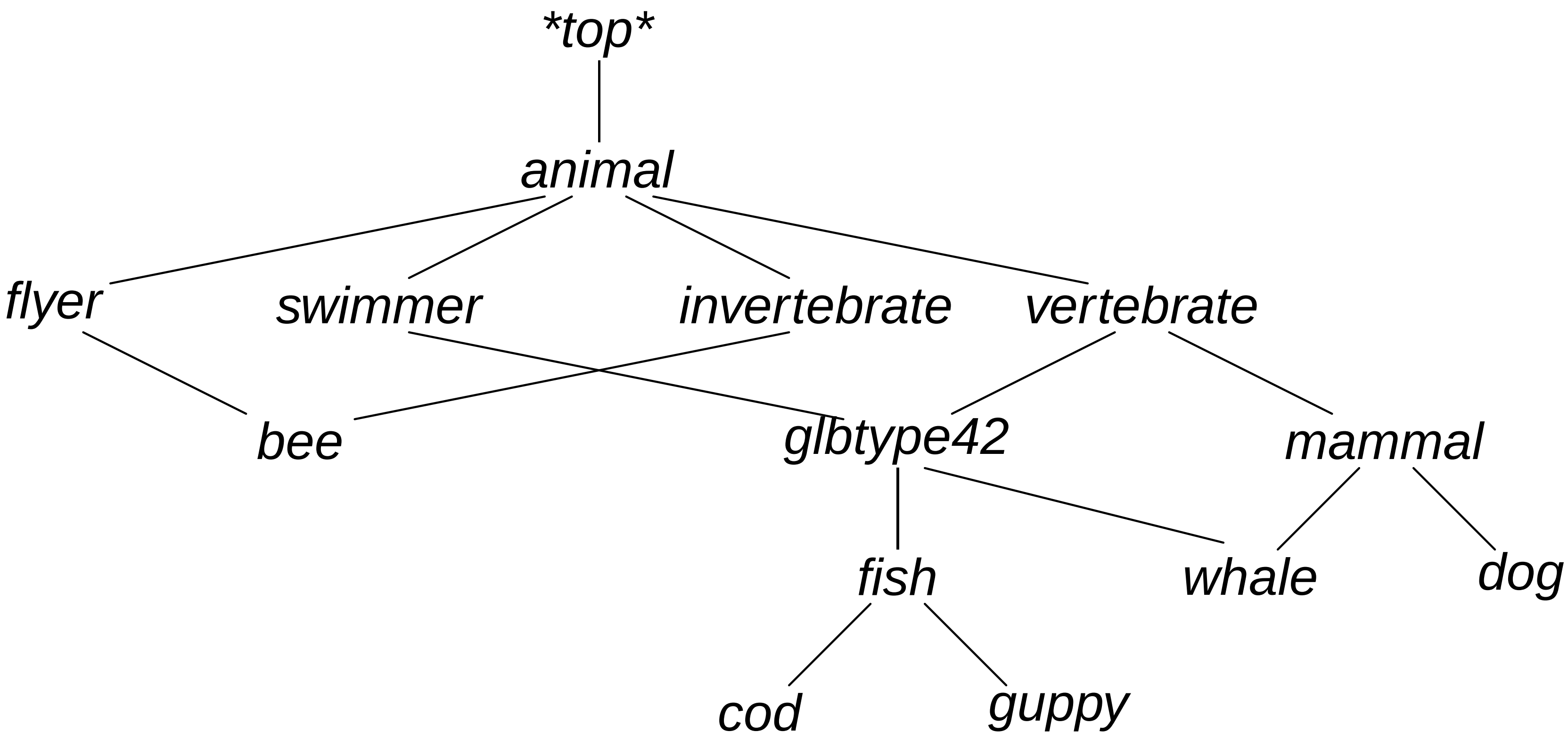
An Invalid Type Hierarchy

- *swimmer* and *vertebrate* have two joint descendants: *fish* and *whale*;
- *fish* and *whale* are incomparable in the hierarchy: glb condition violated.



Fixing the Type Hierarchy

- LKB system introduces *glb types* as required: ‘*swimmer-vertebrate*’.



More Interesting Well-Formed Unification

Type Constraints Associated to *animal* Hierarchy

$$\textit{swimmer} \rightarrow {}_{\textit{swimmer}}\begin{bmatrix}\texttt{FINS}\ \textit{bool}\end{bmatrix} \qquad \textit{mammal} \rightarrow {}_{\textit{mammal}}\begin{bmatrix}\texttt{FRIENDLY}\ \textit{bool}\end{bmatrix}$$

$$\textit{whale} \rightarrow {}_{\textit{whale}}\begin{bmatrix}\texttt{BALEEN} & \textit{bool}\\ \texttt{FINS} & \textit{true}\\ \texttt{FRIENDLY} & \textit{bool}\end{bmatrix}$$

$${}_{\textit{mammal}}\begin{bmatrix}\texttt{FRIENDLY}\ \textit{true}\end{bmatrix} \sqcap {}_{\textit{swimmer}}\begin{bmatrix}\texttt{FINS}\ \textit{bool}\end{bmatrix} \equiv {}_{\textit{whale}}\begin{bmatrix}\texttt{BALEEN} & \textit{bool}\\ \texttt{FINS} & \textit{true}\\ \texttt{FRIENDLY} & \textit{true}\end{bmatrix}$$

$${}_{\textit{mammal}}\begin{bmatrix}\texttt{FRIENDLY}\ \textit{true}\end{bmatrix} \sqcap {}_{\textit{swimmer}}\begin{bmatrix}\texttt{FINS}\ \textit{false}\end{bmatrix} \equiv \bot$$

Recursion in the Type Hierarchy

- Type hierarchy must be finite *after* type inference; illegal type constraint:

```
*list* := *top* & [ FIRST *top*, REST *list* ].
```

- needs additional provision for empty lists; indirect recursion:

```
*list* := *top*.
*ne-list* := *list* & [ FIRST *top*, REST *list* ].
*null* := *list*.
```

- recursive types allow for *parameterized list types* (‘list of X’):

```
*s-list* := *list*.
*s-ne-list* := *ne-list* & *s-list &
               [ FIRST expression, REST *s-list* ].
*s-null* := *null* & *s-list*.
```

Properties of (Our) Type Hierarchies

- **Unique Top** a single hierarchy of all types with a unique top node;
- **No Cycles** no path through the hierarchy from one type to itself;
- **Unique Greatest Lower Bounds** Any two types in the hierarchy are either (a) incompatible (i.e. share no descendants) or (b) have a unique most general ('highest') descendant (called their greatest lower bound);
- **Closed World** all types that exist have a known position in hierarchy;
- **Compatibility** type compatibility in the hierarchy determines feature structure unifiability: two types unify to their greatest lower bound.

Properties of (Our) Typed Feature Structures

- **Finiteness** a typed feature structure has a finite number of nodes;
- **Unique Root and Connectedness** a typed feature structure has a unique root node; apart from the root, all nodes have at least one parent;
- **No Cycles** no node has an arc that points back to the root node or to another node that intervenes between the node itself and the root;
- **Unique Features** any node can have any (finite) number of outgoing arcs, but the arc labels (i.e. features) must be unique within each node;
- **Typing** each node a has single type which is defined in the hierarchy.

The Linguistic Knowledge Builder (LKB)

Compiler and Interactive Debugger

- Grammar definition errors identified at load time by position in file;
- inheritance and appropriateness tracked by type and attributes;
- batch check, expansion, and indexing of full lexicon on demand;
- efficient parser and generator to map between strings and meaning;
- visualization of main data types; interactive stepping and unification.

- Main developers: Copestake (original), Carroll, Malouf, and Oepen;
- implementation: Allegro CL, Macintosh CL, (LispWorks, CMU CL);
- available in open-source and binary form for common platforms.

The Format of Grammar Rules in the LKB

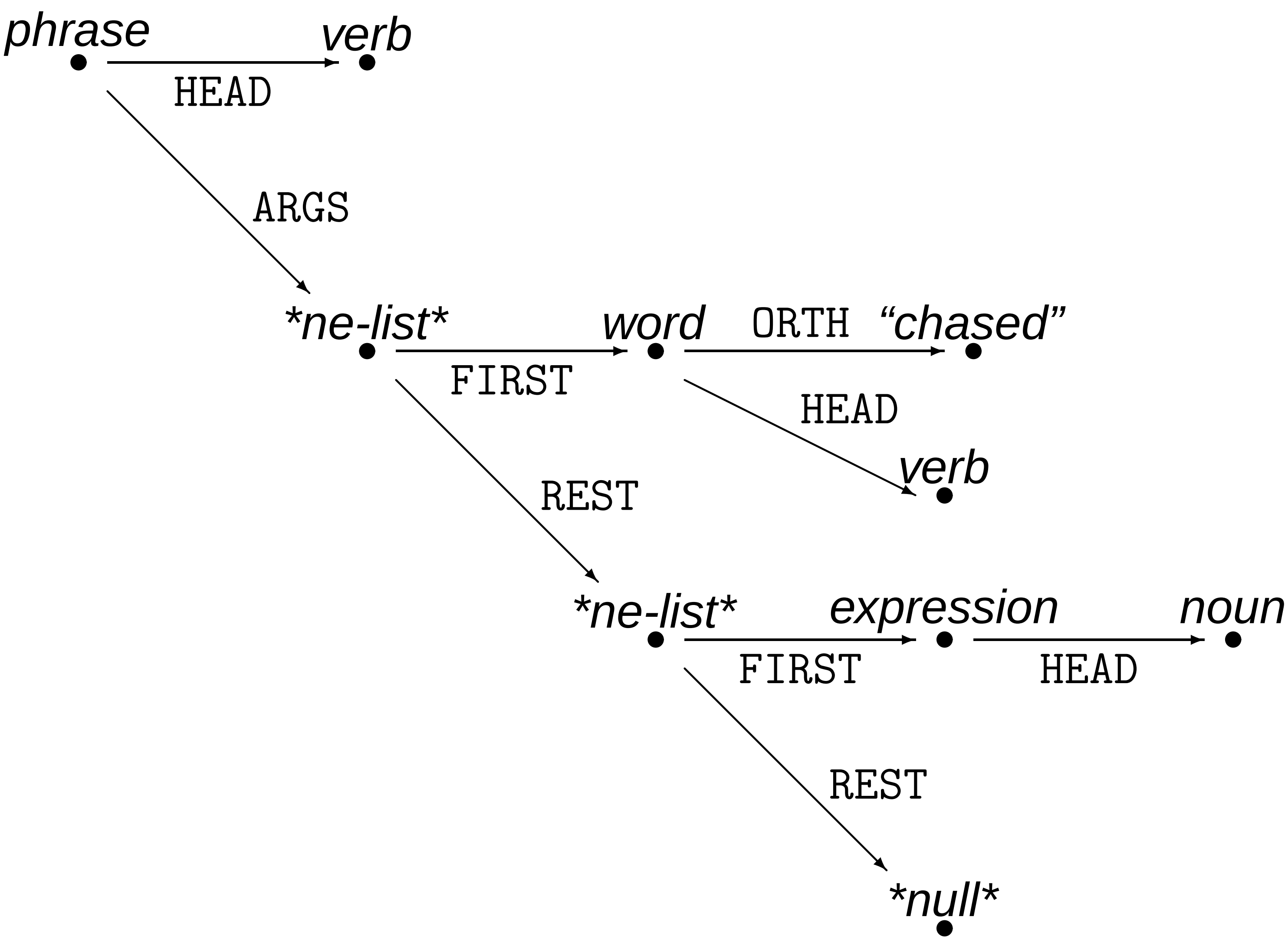
$$
\textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\rangle\\ \text{COMPS} & \boxed{3}\end{bmatrix} \longrightarrow \boxed{2}\,\textit{phrase}\begin{bmatrix}\text{SPR} & \langle\rangle\\ \text{COMPS} & \langle\rangle\end{bmatrix},\quad \textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\boxed{2}\rangle\\ \text{COMPS} & \boxed{3}\end{bmatrix}
$$

$$
\textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\rangle\\ \text{COMPS} & \boxed{3}\\ \text{ARGS} & \left\langle \boxed{2}\,\textit{phrase}\begin{bmatrix}\text{SPR} & \langle\rangle\\ \text{COMPS} & \langle\rangle\end{bmatrix},\ \textit{phrase}\begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SPR} & \langle\boxed{2}\rangle\\ \text{COMPS} & \boxed{3}\end{bmatrix}\right\rangle\end{bmatrix}
$$

Typed Feature Structure Example (as AVM)

$$
\textit{phrase}\begin{bmatrix}
\texttt{HEAD} & \textit{verb} \\
\texttt{ARGS} & \textit{*ne-list*}\begin{bmatrix}
\texttt{FIRST} & \textit{word}\begin{bmatrix}
\texttt{ORTH} & \textit{``chased''} \\
\texttt{HEAD} & \textit{verb}
\end{bmatrix} \\
\texttt{REST} & \textit{*ne-list*}\begin{bmatrix}
\texttt{FIRST} & \textit{expression}\begin{bmatrix}
\texttt{HEAD} & \textit{noun}
\end{bmatrix} \\
\texttt{REST} & \textit{*null*}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

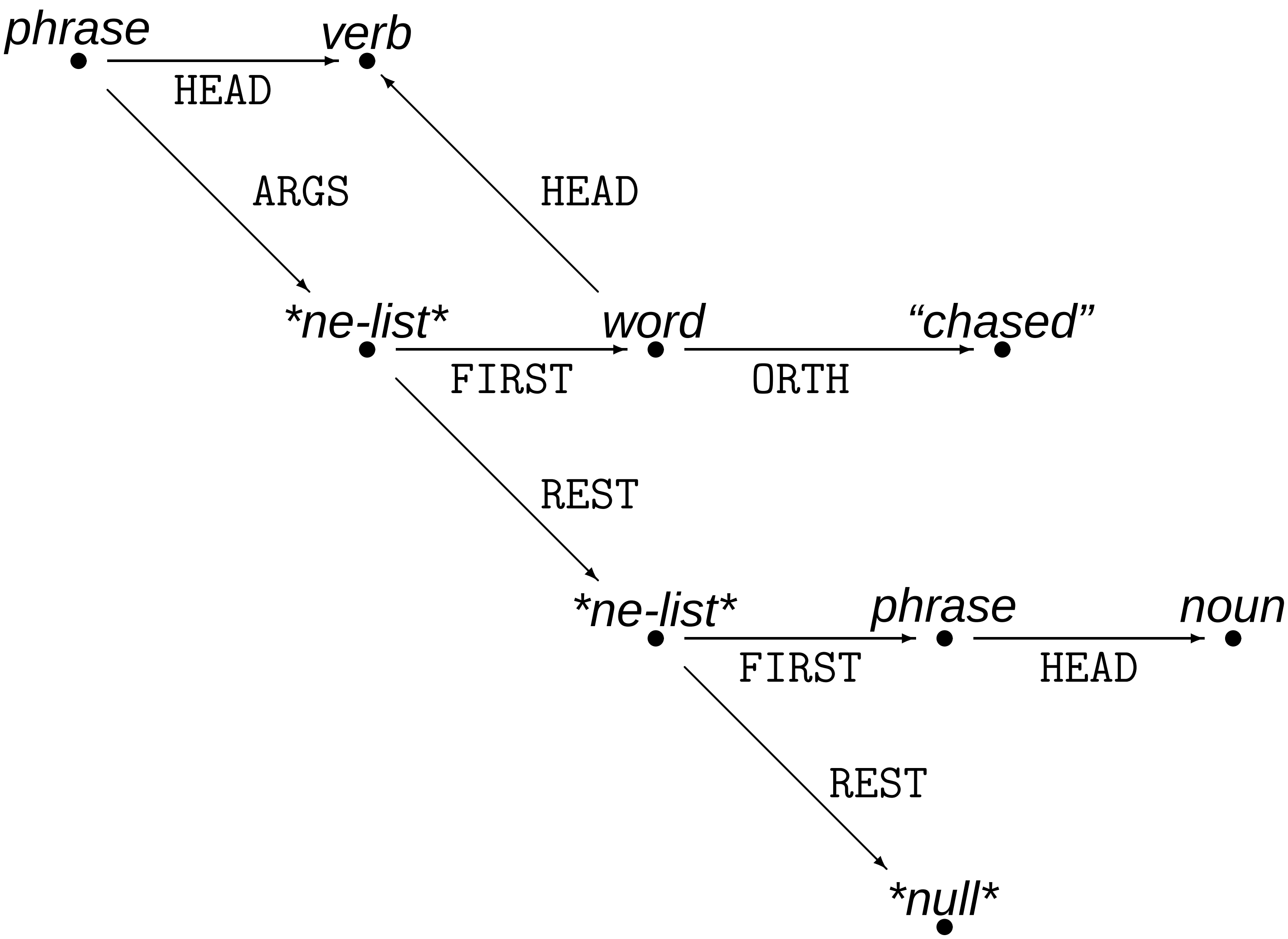
Typed Feature Structure Example (as Graph)



Typed Feature Structure Example (in TDL)

```
vp := phrase &
[ HEAD verb,
  ARGS *ne-list* &
       [ FIRST word &
               [ ORTH "chased",
                 HEAD verb ],
         REST *ne-list* &
              [ FIRST expression &
                      [ HEAD noun ],
                REST *null* ]]] .
```

Reentrancy in a Typed Feature Structure (Graph)



Reentrancy in a Typed Feature Structure (AVM)

```
phrase [ HEAD [1] verb
         ARGS *ne-list* [ FIRST word [ ORTH "chased"
                                       HEAD [1] ]
                          REST *ne-list* [ FIRST phrase [ HEAD noun ]
                                           REST *null* ] ] ]
```

Reentrancy in a Typed Feature Structure (TDL)

```
vp := phrase &
[ HEAD #head & verb,
  ARGS *ne-list* &
       [ FIRST word &
               [ ORTH "chased",
                 HEAD #head ],
         REST *ne-list* &
              [ FIRST phrase &
                      [ HEAD noun ],
                REST *null* ]]] .
```

More Terminology: Grammatical Functions

Licensing — Government — Agreement

The dog barks. — **The dog a cat barks* — **The dog barks a cat.*

Kim depends on Sandy — **Kim depends in Sandy*

The class meets on Thursday in 508 at 12:15.

- **Constituent** node in analysis tree (terminal or instantiation of rule);
- **Head** licenses additional constituents and can govern their form;
- **Specifier** precedes head, singleton, nominative case, agreement;
- **Complement** post-head, licensed and governed, order constraints;
- **Adjunct** ‘free’ modifier, optional, may iterate, designated position;
- **Government** directed: a property of c_1 determines the form of c_2;
- **Agreement** bi-directional: co-occurence of properties on c_1 and c_2.

An Ambiguous Example

Kim shoveled snow on lifts.

A Highly Ambiguous Example

The manager placed his bid on my desk.