# Topics in Computational Linguistics — Grammar Engineering —

**Dan Flickinger**

CSLI Stanford & Saarland University

danf@csli.stanford.edu

**Stephan Oepen**

Universitetet i Oslo & CSLI Stanford

oe@csli.stanford.edu

http://lingo.stanford.edu/courses/05/ge/

# The Linguistic Knowledge Builder (LKB)

## General & History

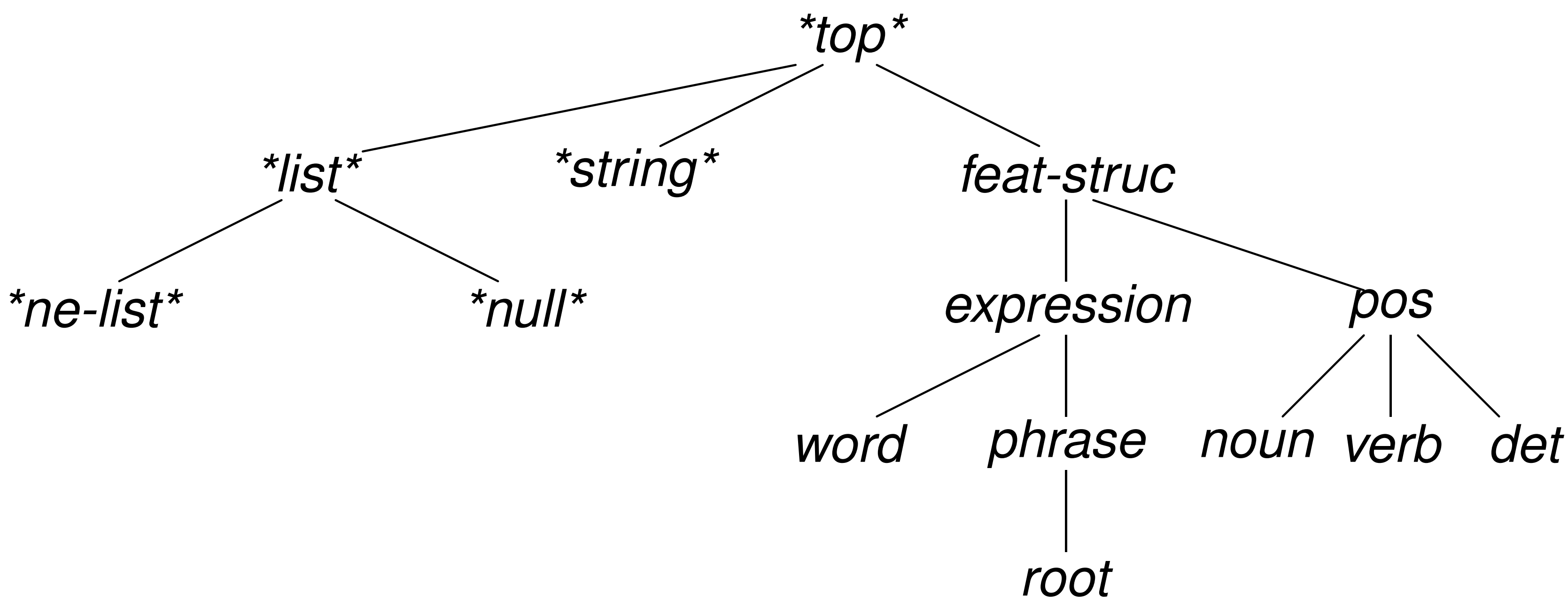
- Specialized grammar engineering environment for TFS grammars;
- main developers: Copestake (original), Carroll, Malouf, and Oepen;
- open-source and binary distributions (Linux, Windows, and Solaris).

## Grammar Engineering Fuctionality

- Compiler for typed feature structure grammars → wellformedness;
- parser and generator: map from strings to meaning and vice versa;
- visualization: inspect trees, feature structures, intermediate results;
- debugging and tracing: interactive unification, ‘stepping’, et al.

# The Type Hierarchy: Fundamentals

- Types ‘represent’ groups of entities with similar properties (‘classes’);
- types ordered by specificity: subtypes inherit properties of (all) parents;
- type hierarchy determines which types are compatible (and which not).

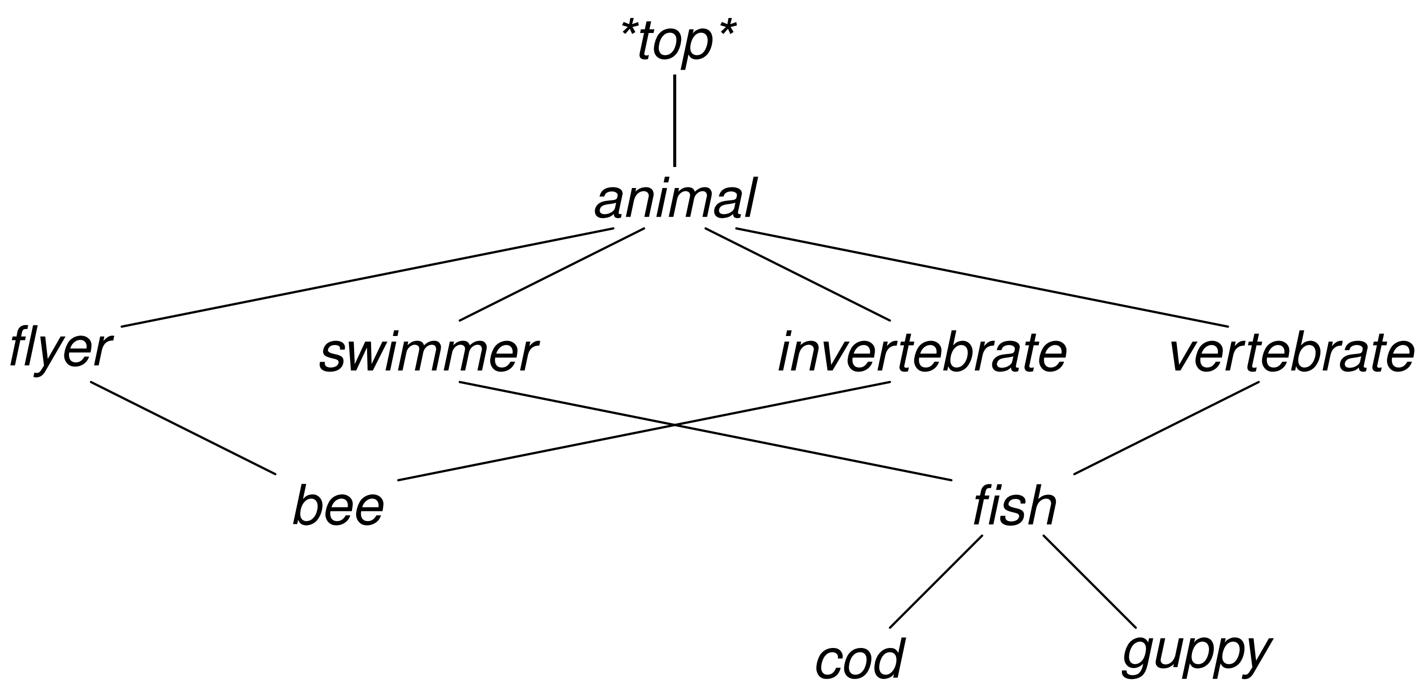


# Properties of (Our) Type Hierarchies

- **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.

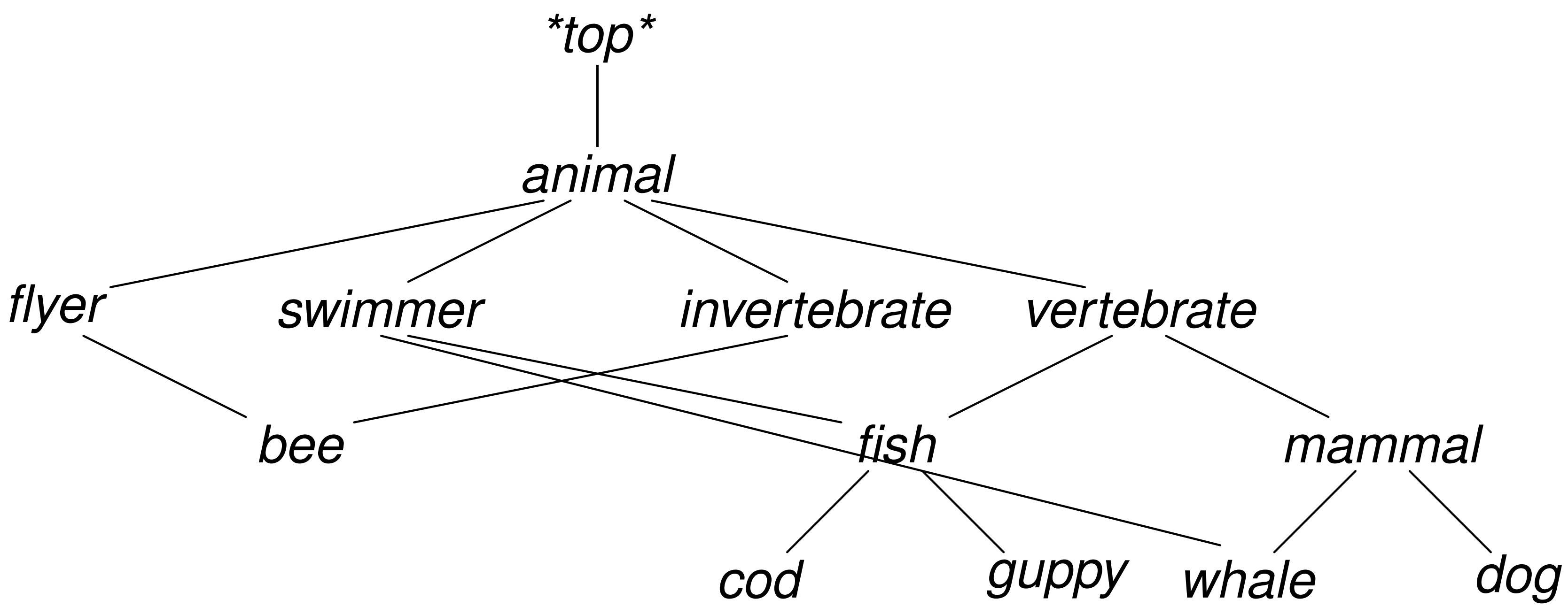
# Multiple Inheritance

- *flyer* and *swimmer* no common descendants: they are incompatible;
- *flyer* and *bee* stand in hierarchical relationship: they unify to subtype;
- *flyer* and *invertebrate* have a unique greatest common descendant.



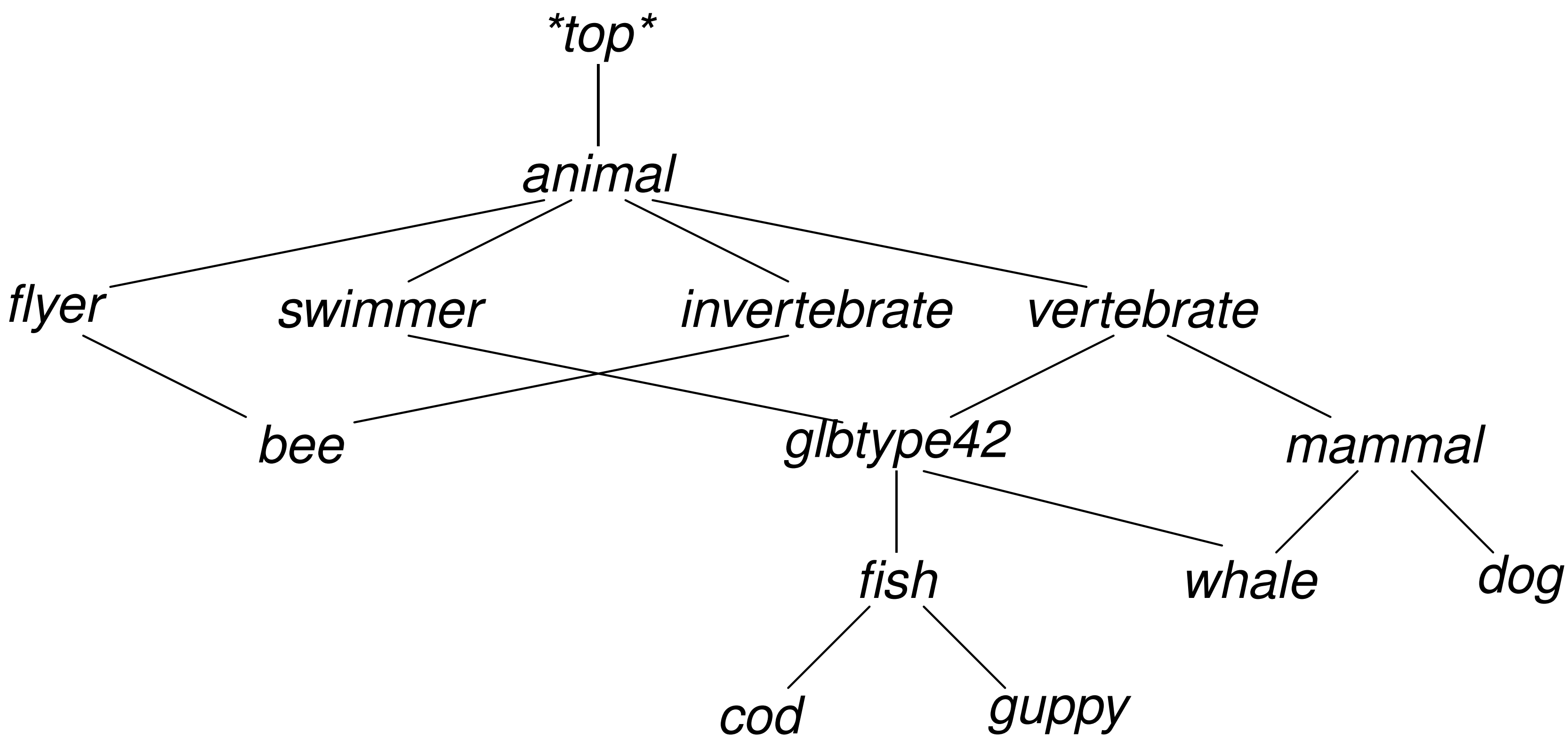
# An Invalid Type Hierarchy

- *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*’.



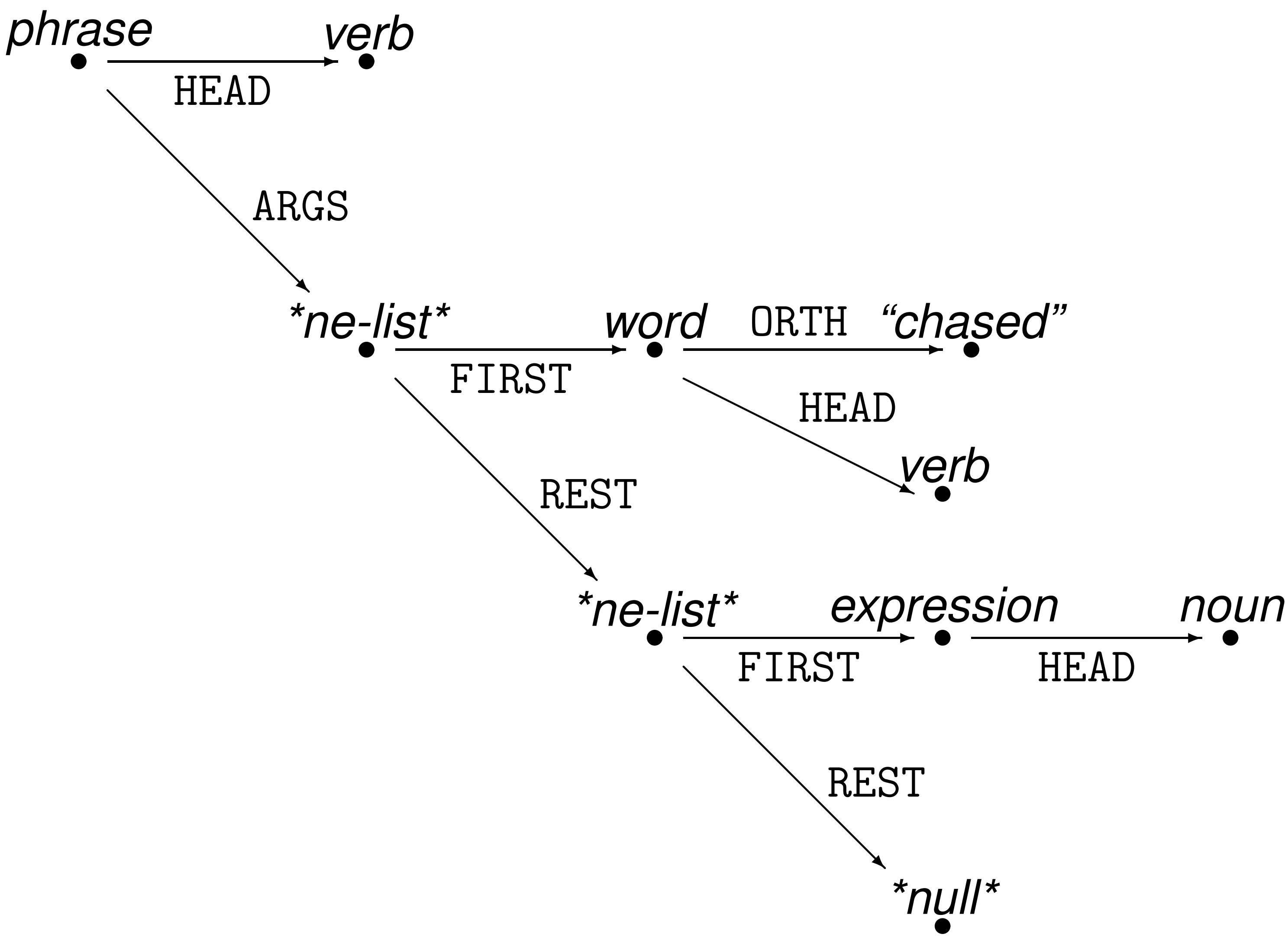
# Properties of Typed Feature Structures

- **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 has single type which is defined in the hierarchy.

# Typed Feature Structure Example (as AVM)

$$
{}_{\textit{phrase}}\left[\begin{array}{ll}
\texttt{HEAD} & \textit{verb} \\
\texttt{ARGS} & {}_{\textit{*ne-list*}}\left[\begin{array}{ll}
\texttt{FIRST} & {}_{\textit{word}}\left[\begin{array}{ll}
\texttt{ORTH} & \textit{“chased”} \\
\texttt{HEAD} & \textit{verb}
\end{array}\right] \\
\texttt{REST} & {}_{\textit{*ne-list*}}\left[\begin{array}{ll}
\texttt{FIRST} & {}_{\textit{expression}}\left[\begin{array}{ll}
\texttt{HEAD} & \textit{noun}
\end{array}\right] \\
\texttt{REST} & \textit{*null*}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

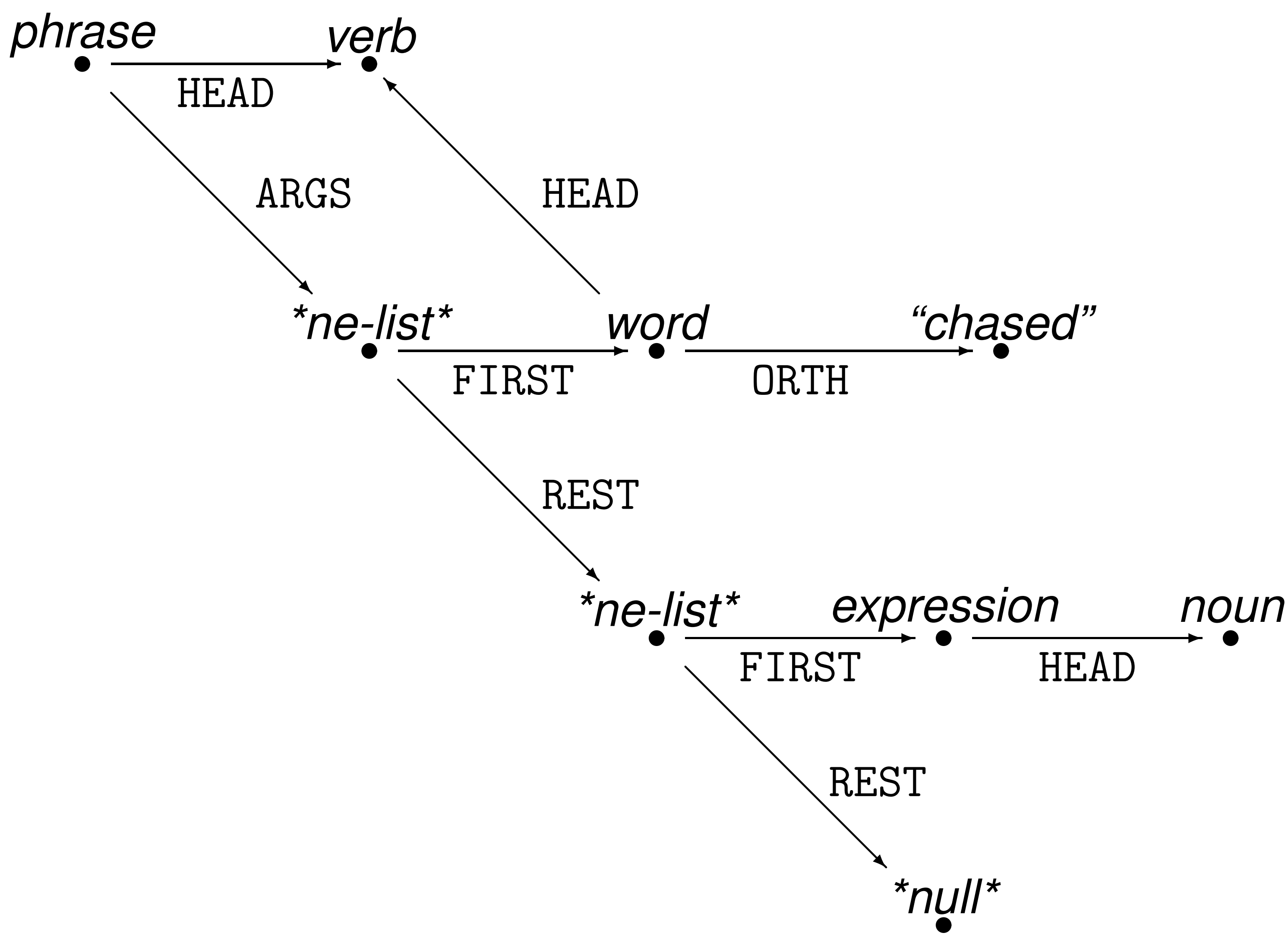
# Typed Feature Structure Example (as Graph)



# 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 *expression*
      - 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 expression &
                      [ HEAD noun ],
                REST *null* ]]] .
```

# Typed Feature Structure Subsumption

- Typed feature structures can be partially ordered by information content;
- a more general structure is said to *subsume* a more specific one;
- $_{\textit{*top*}}\left[\,\right]$ is the most general feature structure (while $\bot$ is inconsistent);
- $\sqsubseteq$ ('square subset or equal') conventionally used to depict subsumption.

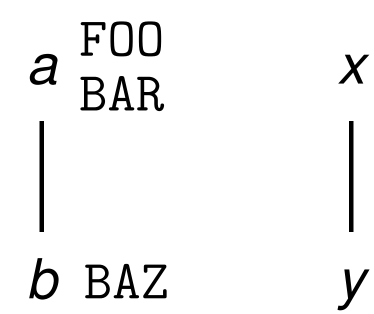
Feature structure $F$ subsumes feature structure $G$ ($F \sqsubseteq G$) iff: (1) if path $p$ is defined in $F$ then $p$ is also defined in $G$ and the type of the value of $p$ in $F$ is a supertype or equal to the type of the value of $p$ in $G$, and (2) all paths that are reentrant in $F$ are also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

# Feature Structure Subsumption: Examples

$$\mathrm{TFS}_1: \; {}_{a}\begin{bmatrix} \texttt{FOO} & x \\ \texttt{BAR} & x \end{bmatrix} \qquad \mathrm{TFS}_2: \; {}_{a}\begin{bmatrix} \texttt{FOO} & x \\ \texttt{BAR} & y \end{bmatrix}$$

$$\mathrm{TFS}_3: \; {}_{b}\begin{bmatrix} \texttt{FOO} & y \\ \texttt{BAR} & x \\ \texttt{BAZ} & x \end{bmatrix} \qquad \mathrm{TFS}_4: \; {}_{a}\begin{bmatrix} \texttt{FOO} & \boxed{1}\, x \\ \texttt{BAR} & \boxed{1} \end{bmatrix}$$

**Signature**




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

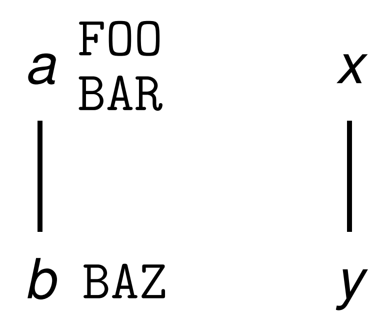
- Decide whether two typed feature structures are mutually compatible;
- determine combination of two TFSs to give the most general feature structure which retains all information which they individually contain;
- if there is no such feature structure, unification fails (depicted as $\bot$);
- unification *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

$\text{TFS}_1\!: \; {}_{a}\!\begin{bmatrix} \text{FOO } x \\ \text{BAR } x \end{bmatrix}$ $\qquad$ $\text{TFS}_2\!: \; {}_{a}\!\begin{bmatrix} \text{FOO } x \\ \text{BAR } y \end{bmatrix}$

$\text{TFS}_3\!: \; {}_{b}\!\begin{bmatrix} \text{FOO } y \\ \text{BAR } x \\ \text{BAZ } x \end{bmatrix}$ $\qquad$ $\text{TFS}_4\!: \; {}_{a}\!\begin{bmatrix} \text{FOO } \boxed{1}\, x \\ \text{BAR } \boxed{1} \end{bmatrix}$

## Signature




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

# 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** | **ne-list** [FIRST **top**, REST **list**] | FIRST and REST |

# Type Inference: Making a TFS Well-Formed

- Apply all type constraints to convert a TFS into a well-formed TFS;
- determine most general well-formed TFS subsumed by the input TFS;
- specialize all types so that all features are appropriate:

$$\textit{*top*}\begin{bmatrix} \texttt{HEAD} & \textit{pos} \\ \texttt{ARGS} & \textit{*list*} \end{bmatrix} \longrightarrow \textit{phrase}\begin{bmatrix} \texttt{HEAD} & \textit{pos} \\ \texttt{ARGS} & \textit{*list*} \end{bmatrix}$$

- expand all nodes with the type constraint of the type of that node:

$$\textit{phrase}\begin{bmatrix} \texttt{HEAD} & \textit{pos} \\ \texttt{ARGS} & \textit{*list*} \end{bmatrix} \longrightarrow \textit{phrase}\begin{bmatrix} \texttt{HEAD} & \textit{pos} \\ \texttt{ARGS} & \textit{*list*} \\ \texttt{SPR} & \textit{*list*} \\ \texttt{COMPS} & \textit{*list*} \end{bmatrix}$$

# More Interesting Well-Formed Unification

**Type Constraints Associated to Earlier *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*.
```

# Notational Conventions

- lists not available as built-in data type; abbreviatory notation in TDL:

```
< a, b > ≡ [ FIRST a, REST [ FIRST b, REST *null* ] ]
```

- underspecified (variable-length) list:

```
< a ... > ≡ [ FIRST a, REST *list* ]
```

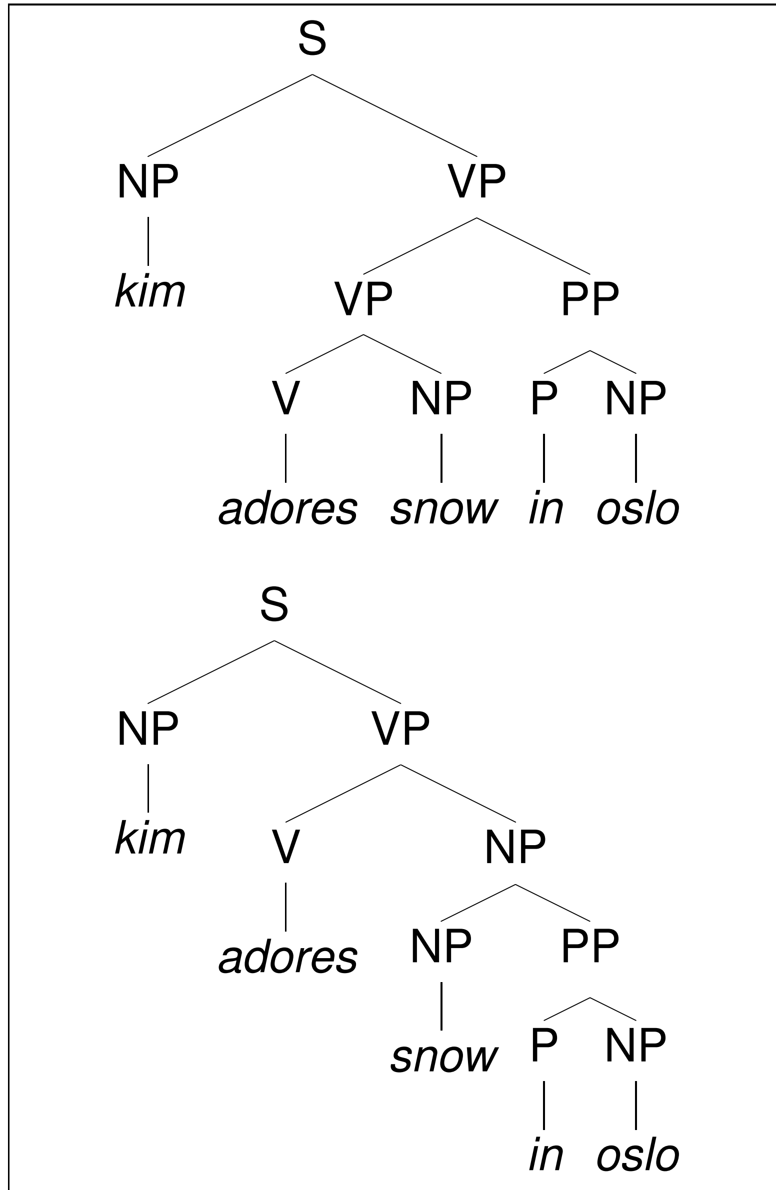
- difference (open-ended) lists; allow concatenation by unification:

```
<! a !> ≡ [ LIST [ FIRST a, REST #tail ], LAST #tail ]
```

- built-in and ‘non-linguistic’ types pre- and suffixed by asterisk (**top**);
- strings (e.g. *“chased”*) need no declaration; always subtypes of **string**;
- strings cannot have subtypes and are (thus) mutually incompatible.

# Recognizing the Language of a Grammar

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
VP $\rightarrow$ VP PP
NP $\rightarrow$ NP PP
PP $\rightarrow$ P NP
NP $\rightarrow$ kim | snow | oslo
V $\rightarrow$ snores | adores
P $\rightarrow$ in




**All Complete Derivations**

- are rooted in the start symbol $S$;
- label internal nodes with categories $\in C$, leafs with words $\in \Sigma$;
- instantiate a grammar rule $\in P$ at each local subtree of depth one.

# Structured Categories in a Unification Grammar

- All (constituent) categories in the grammar are typed feature structures;
- specific TFS configurations may correspond to 'traditional' categories;
- labels like 'S' or 'NP' are mere abbreviations, not elements of the theory.

$$\mathit{word}\begin{bmatrix} \texttt{HEAD} & \mathit{noun} \\ \texttt{SPR} & \langle [\;] \rangle \\ \texttt{COMPS} & \langle \rangle \end{bmatrix} \qquad \mathit{phrase}\begin{bmatrix} \texttt{HEAD} & \mathit{verb} \\ \texttt{SPR} & \langle \rangle \\ \texttt{COMPS} & \langle \rangle \end{bmatrix} \qquad \mathit{phrase}\begin{bmatrix} \texttt{HEAD} & \mathit{verb} \\ \texttt{SPR} & \langle [\;] \rangle \\ \texttt{COMPS} & \langle \rangle \end{bmatrix}$$

| 'N' | 'S' | 'VP' |
|---|---|---|
| **'lexical'** | **'maximal'** | **'intermediate'** |

# The Format of Grammar Rules in the LKB

$$
\textit{mother}\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \boxed{2} \\ \texttt{COMPS} & \langle\,\rangle \\ \ldots & \end{bmatrix} \longrightarrow \textit{daughter}_1\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \boxed{2} \\ \texttt{COMPS} & \langle\boxed{3}\rangle \end{bmatrix}, \quad \textit{daughter}_2\;\boxed{3}\begin{bmatrix} \ldots \end{bmatrix}
$$

$$
\textit{mother}\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \boxed{2} \\ \texttt{COMPS} & \langle\,\rangle \\ \ldots & \\ \texttt{ARGS} & \left\langle \textit{daughter}_1\begin{bmatrix}\texttt{HEAD} & \boxed{1} \\ \texttt{SPR} & \boxed{2} \\ \texttt{COMPS} & \langle\boxed{3}\rangle \end{bmatrix}, \quad \textit{daughter}_2\;\boxed{3}\begin{bmatrix} \ldots \end{bmatrix} \right\rangle \end{bmatrix}
$$

# Our Grammars: Table of Contents

## Type Description Language (TDL)

- `types.tdl` type definitions: hierarchy of grammatical knowledge;
- `lexicon.tdl` instances of (lexical) types plus orthography;
- `rules.tdl` instances of construction types; used by the parser;
- `lrules.tdl` lexical rules, applied before non-lexical rules;
- `irules.tdl` lexical rules that require orthographemic variation.

## Auxiliary Files (Grammar Configuration for LKB)

- `globals.lsp.` Parameter settings (e.g. path to orthography et al.);
- `user-fns.lsp` (small number) of LKB interface functions;
- `mrsglobals.lsp` MRS parameters (path to semantics et al.)