LKB Formalism
Lab 1 questions

Ling 567
January 6, 2010

Overview

- Type hierarchies, inheritance, unification
- Typed feature structures, subsumption, unification
- Type constraints, making typed feature structures well-formed
- Notational conventions
- Grammar rules in the LKB
- Lab 1 questions

tdl and typed feature structures

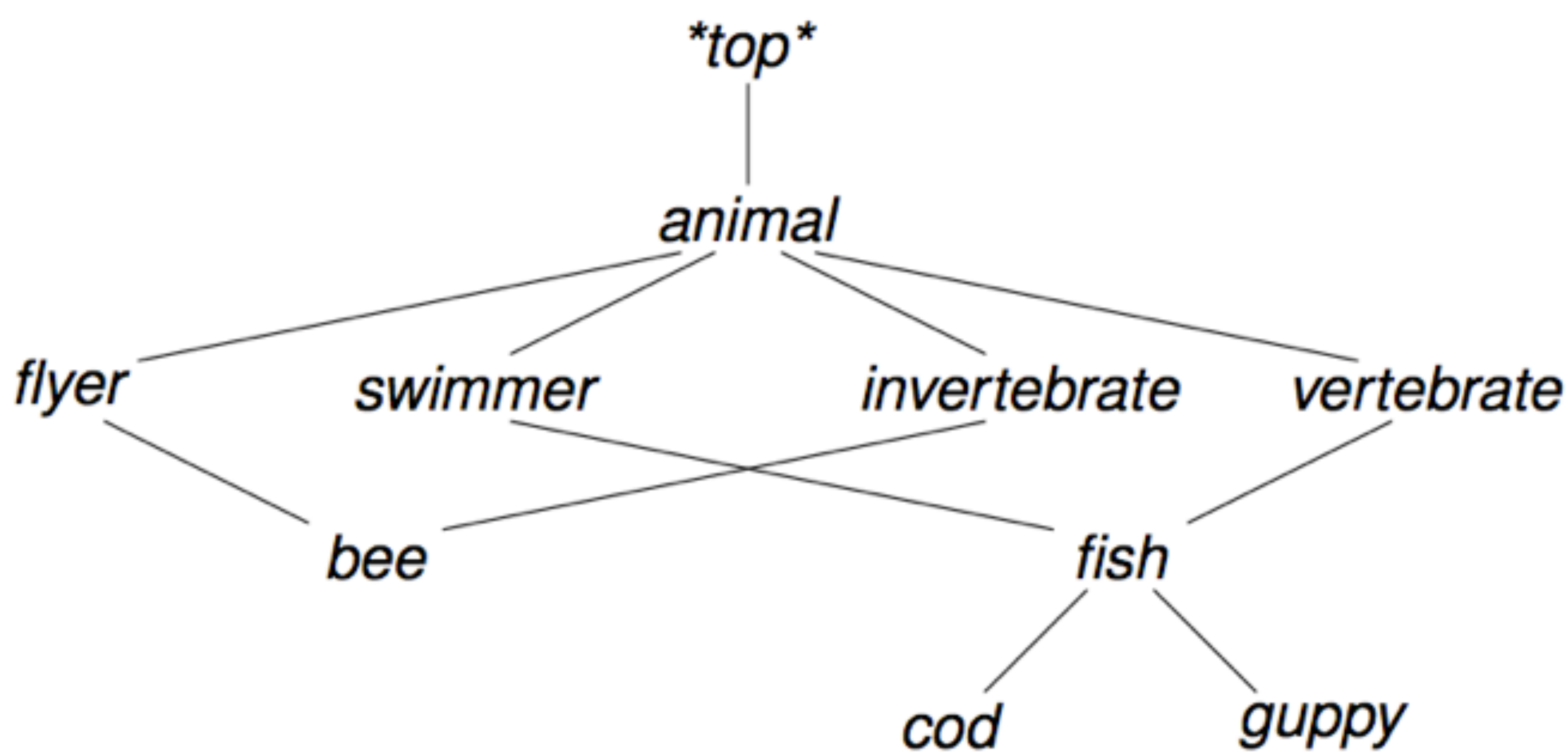
- tdl = type description language
- .tdl files encode type *descriptions.*
- The LKB reads in the tdl files and compiles the type descriptions into a well-formed type hierarchy.
- NB: Actual trees are not subject to the constraint that they be fully specified, but they must be well-typed (all features appropriate for a type are present, though types need not be maximally specific).

Properties of our type hierarchies

- Unique top: All types ultimately inherit from one top node
- No cycles: No path through the hierarchy from a type to itself
- Unique greatest lower bounds (glbs): Any two types in the hierarchy are either incompatible (share no descendants) or have a unique most general subtype
- Closed world: All types that exist have a known position in the hierarchy
- Compatibility: Two compatible types unify to their glb

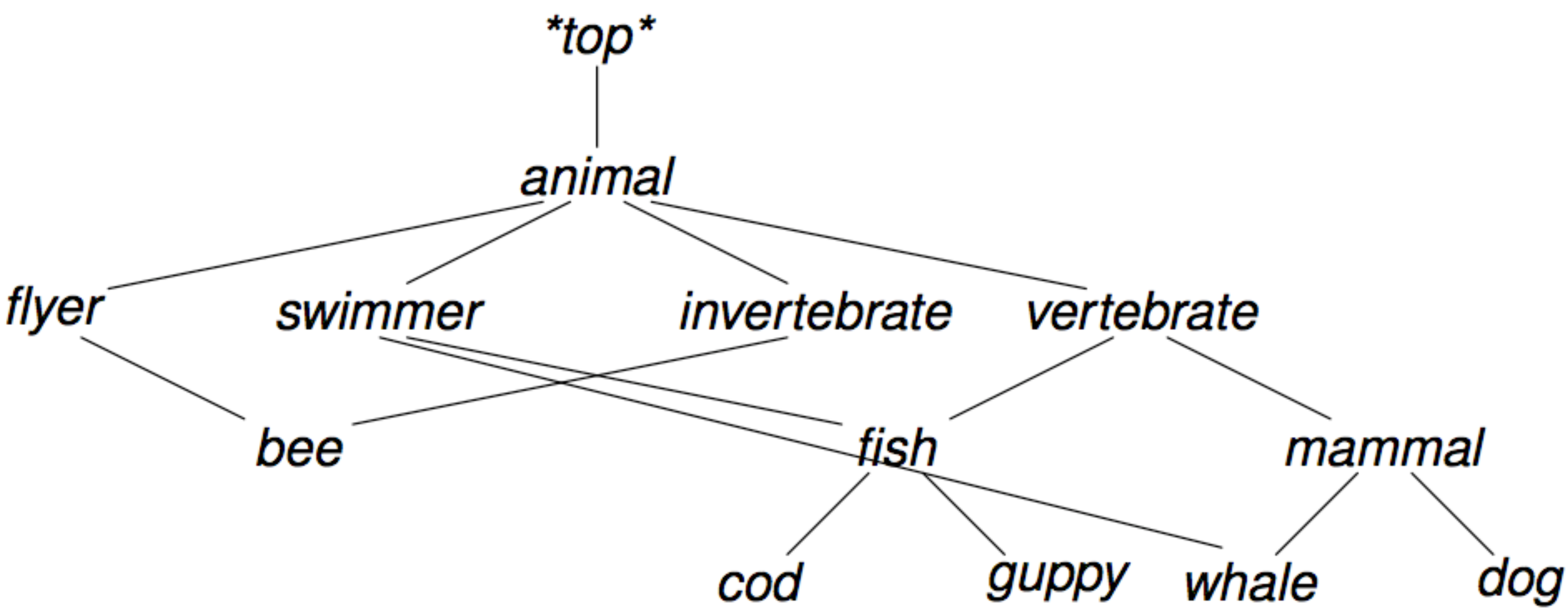
Multiple inheritance and unification

- *flyer* and *swimmer* are incompatible (no common descedants)
- *flyer* and *bee* unify to subtype (hierarchical relationship)
- *flyer* and *invertebrate* unify to glb (*bee)*



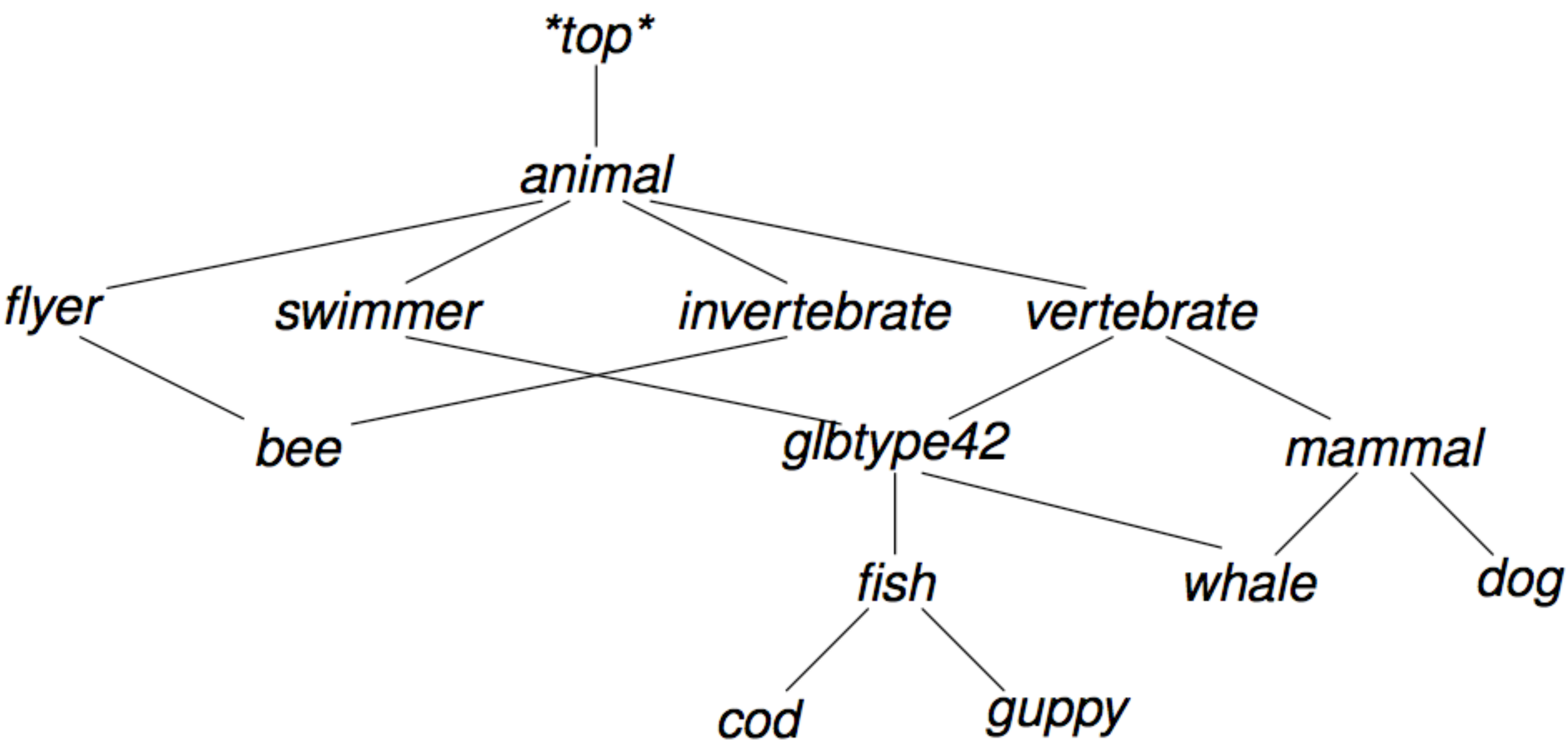
An invalid type hierarchy

- *swimmer* and *invertebrate* have two common subtypes: *fish* and *whale*
- *fish* and *whale* are incomparable in the hierarchy: glb condition is violated



Fixing the type hierachy

- The LKB introduces glb types as required



Properties of typed feature structures

- Finiteness: A typed feature structure has a finite number of nodes
- Unique root and connectedness: A tfs has a unique root parent; all other 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 any (finite) number of outgoing arcs, but the arc labels (i.e., features) must be unique within each node
- Typing: Each node has a single type which is defined in the hierarchy

tdl example

```
type := supertype1 & supertype2 &
  [ FEAT1 val1,
    FEAT2 val2 & [ FEAT3 #same,
                   FEAT4 #same ] ].
```

Typed feature structure subsumption

- tfss can be partially ordered by information content
- a more general structure is said to *subsume* a more specific one
- *top* is the most general feature structure, while $\perp$ is inconsistent
- Feature structure *F* subsumes feature structure *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 as supertype or equal to the value of *p* in *G*, and (2) all paths that are reentrant in *F* are also reentrant in *G.*

Subsumption examples

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

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

Signature

$$\begin{array}{ll} a & \begin{array}{l}\text{FOO}\\ \text{BAR}\end{array} \\ | & \\ b & \text{BAZ} \end{array} \qquad\qquad \begin{array}{c} x \\ | \\ y \end{array}$$

Which tfss subsume which other tfss?

Typed Feature Structure Unification

- Decide whether the two typed feature structures are compatible

- Determine the combination of the two tfss which gives the most general feature structure which retains all of the information they each individually contain

- Unification *monotonically* combines information from both ‘input’ tfss

- The unification of *F* and *G* is the most general tfs that is subsumed by both *F* and *G* (if it exists).

Unification examples

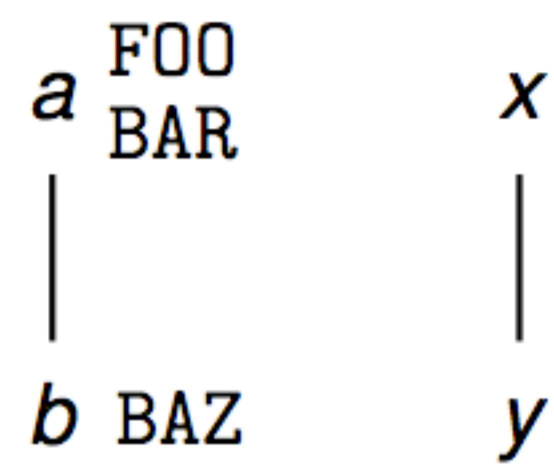
TFS$_1$: $\left[{}_a \begin{array}{l} \text{FOO } x \\ \text{BAR } x \end{array}\right]$

TFS$_2$: $\left[{}_a \begin{array}{l} \text{FOO } x \\ \text{BAR } y \end{array}\right]$

TFS$_3$: $\left[{}_b \begin{array}{l} \text{FOO } y \\ \text{BAR } x \\ \text{BAZ } x \end{array}\right]$

TFS$_4$: $\left[{}_a \begin{array}{l} \text{FOO } \boxed{1}\, x \\ \text{BAR } \boxed{1} \end{array}\right]$

Signature



What is the unification of TFS1&2?
1&3? 3&4?

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 its *appropriate features*

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

Type inference: Making a tfs well-formed

- Apply all type constraints to convert tfs to well-formed tfs
- Determine most general well-formed tfs subsumed by input tfs
- Specialize all types so that all features are appropriate
- Expand all nodes with the type constraint of the type on that node

Examples

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

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

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 hierachy

- 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*:

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

Notational conventions

- Lists are not available as a 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 ]
```

Notational conventions

- strings (e.g., “chased”) need no declaration; they are always subtypes of *string*
- strings cannot have subtypes, and are (thus) mutually incompatible

Format of grammar rules in the LKB

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

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

Overview

- Type hierarchies, inheritance, unification
- Typed feature structures, subsumption, unification
- Type constraints, making typed feature structures well-formed
- Notational conventions
- Grammar rules in the LKB
- Lab 1 questions