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 descendents)
- *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 hierarchy

- 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
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

- \textit{top} is the most general feature structure, while \( \bot \) 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

TFS₁: \[
\begin{bmatrix}
  \text{FOO} & x \\
  \text{BAR} & x
\end{bmatrix}
\]

TFS₂: \[
\begin{bmatrix}
  \text{FOO} & x \\
  \text{BAR} & y
\end{bmatrix}
\]

TFS₃: \[
\begin{bmatrix}
  \text{FOO} & y \\
  \text{BAR} & x \\
  \text{BAZ} & x
\end{bmatrix}
\]

TFS₄: \[
\begin{bmatrix}
  \text{FOO} & 1 & x \\
  \text{BAR} & 1 & x
\end{bmatrix}
\]

Signature

\[
\begin{array}{ccc}
  a & \text{FOO} & x \\
  b & \text{BAZ} & y
\end{array}
\]

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 tfss that is subsumed by both $F$ and $G$ (if it exists).
Unification examples

What is the unification of TFS1&2? TFS3&2? TFS3&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*

<table>
<thead>
<tr>
<th>type</th>
<th>constraint</th>
<th>appropriate features</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ne-list</em></td>
<td>FIRST</td>
<td><em>top</em></td>
</tr>
</tbody>
</table>
|          | REST   | *list* | 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

\[
\begin{align*}
\text{HEAD} & \quad \text{pos} \\
\text{ARGS} & \quad \text{*list*} \\
\end{align*}
\rightarrow
\begin{align*}
\text{phrase} & \\
\text{HEAD} & \quad \text{pos} \\
\text{ARGS} & \quad \text{*list*} \\
\end{align*}
\]

\[
\begin{align*}
\text{phrase} & \\
\text{HEAD} & \quad \text{pos} \\
\text{ARGS} & \quad \text{*list*} \\
\end{align*}
\rightarrow
\begin{align*}
\text{phrase} & \\
\text{HEAD} & \quad \text{pos} \\
\text{ARGS} & \quad \text{*list*} \\
\text{SPR} & \\
\text{COMPS} & \quad \text{*list*} \\
\end{align*}
\]
More interesting well-formed unification

Type Constraints Associated to Earlier animal Hierarchy

\[
\begin{align*}
\text{swimmer} & \rightarrow \quad \text{swimmer} \left[ \text{FINS bool} \right] \\
\text{mammal} & \rightarrow \quad \text{mammal} \left[ \text{FRIENDLY bool} \right] \\
\text{whale} & \rightarrow \quad \text{whale} \left[ \text{BALEEN bool} \right. \\
& \quad \left. \text{FINS true} \right. \\
& \quad \left. \text{FRIENDLY bool} \right]
\end{align*}
\]

\[
\begin{align*}
\text{mammal} \left[ \text{FRIENDLY true} \right] \sqcap \text{swimmer} \left[ \text{FINS bool} \right] & \equiv \quad \text{whale} \left[ \text{BALEEN bool} \right. \\
& \quad \left. \text{FINS true} \right. \\
& \quad \left. \text{FRIENDLY true} \right]
\end{align*}
\]

\[
\begin{align*}
\text{mammal} \left[ \text{FRIENDLY true} \right] \sqcap \text{swimmer} \left[ \text{FINS false} \right] & \equiv \perp
\end{align*}
\]
Recursion in the type hierarchy

- Type hierarchy must be finite after type inference; illegal type constraint:
  
  \[ \text{*list*} := \text{*top*} \& \left[ \text{FIRST} \text{*top*}, \text{REST} \text{*list*} \right]. \]

- Needs additional provision for empty lists; indirect recursion:
  
  \[ \text{*list*} := \text{*top*}. \]
  
  \[ \text{*ne-list*} := \text{*list*} \& \left[ \text{FIRST} \text{*top*}, \text{REST} \text{*list*} \right]. \]
  
  \[ \text{*null*} := \text{*list*}. \]

- Recursive types allow for parameterized list types:
  
  \[ \text{*s-list*} := \text{*top*}. \]
  
  \[ \text{*s-ne-list*} := \text{*ne-list*} \& \text{*s-list*} \& \]
  
  \[ \left[ \text{FIRST} \text{*top*}, \text{REST} \text{*list*} \right]. \]
  
  \[ \text{*s-null*} := \text{*list*} \& \text{*s-list*}. \]
Notational conventions

• Lists are not available as a built-in data type; abbreviatory notation in tdl:

\[
< a, b > \equiv [ \text{FIRST } a, \text{REST } [ \text{FIRST } b, \text{REST } *\text{null}* ] ]
\]

• Underspecified (variable-length) list:

\[
< a \ldots > \equiv [ \text{FIRST } a, \text{REST } *\text{list}* ]
\]

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

\[
<! a !> \equiv [ \text{LIST } [ \text{FIRST } a, \text{REST } \#\text{tail } ], \text{LAST } \#\text{tail } ]
\]
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
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