Defaults and lexical prototypes

Workshop on defaults in morphological theory
May 21, 2012

Rob Malouf
San Diego State University
Head-Driven Phrase Structure

Representations in HPSG are **typed feature structures**, a class of directed acyclic graphs.

An **attribute value matrix** is a description which picks out a set of these linguistic objects.

Each feature structure has a **type** associated with it.

Types are organized into a **signature** which specifies appropriateness and inheritance relationships.
Types are organized into an inheritance hierarchy, an ontology of object types.

The hierarchy is a bounded complete partial order: every pair of types have a unique least upper bound and there is a unique most-general-type.
Sort hierarchies

The inheritance hierarchy defines an ontology of linguistic objects (sorts):

- types and their relations (‘is a’ and ‘has a’)
- appropriate features
- appropriate values
- type inference

Provides a basis for precise and efficient implementation (Flickinger 2000)

This ontology is (mostly) arbitrary and (mostly) universal

This metalanguage is important but not by itself linguistically very interesting
Sort hierarchies

Grammar Matrix (Bender, et al. 2010)
Sort hierarchies

Grammar Matrix (Bender, et al. 2010)

\[
\begin{array}{c}
\text{avm} \\
\text{sign-min} \\
\text{STEM} \\
\text{list} \\
\text{basic-sign} \\
\text{KEY-ARG} \\
\text{bool} \\
\text{sign} \\
\text{SYNSEM} \\
\text{ARGS} \\
\text{INFLECTED} \\
\text{synsem} \\
\text{list} \\
\text{inflected}
\end{array}
\]
Lexical hierarchies

The type hierarchy is also used to define constraints on the lexicon and the inventory of constructions

Classes of words can be the same in some ways and different in others

Patterns of sameness can be reified as super-types, while differences are instantiated on lower types in the hierarchy

 Anything that is true of a type is also true of all of any more specific type

Taxonomic approach to linguistic description
Lexical hierarchies

This style of representation associates patterns of sameness and differentness with particular types.


<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>+</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>b</td>
<td>+</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>c</td>
<td>–</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

```
thing

[H -] [G +] [F +] [F -] [H +] [G -]
```

```
a  c  b
```

Tuesday, May 22, 12
**Default inheritance**

**Default** constraints offer a solution to this problem

We can state properties of a type which usually hold, but allow more specific subtypes to override that

Anything that is true of a type is also true of all of any more specific type unless there’s a conflict

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>+</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>b</td>
<td>+</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>c</td>
<td>–</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

```
thing
F +
G +
H +
```

```
[ a ]
H –
[ b ]
G –
[ c ]
F –
```
Default inheritance

Defaults give us a mechanism for representing prototypes

Once we allow overriding, what does it mean to be a member of a category?

Two mechanisms for capturing similarities and differences
Prototypes

Inheritance hierarchies (with or without overriding) come from the same knowledge representation tradition as object-oriented programming.

**Prototype-based** programming is an alternative that has been gaining interest (Borning 1986, Lieberman 1986, Ungar and Smith 1987)

- No abstract classes, only fully specified objects
- All constraints are defaults
- New objects are defined *differentially*
- Objects are related to other objects via *delegation*
Prototypes

Inheritance

reflects an ‘is-a’ relation: a transitive verb is a kind of verb

default overriding is exceptionality

intensional classes and abstract prototypes

Delegation

reflects and ‘is-like’ relation: the lexical entry for *walk* is similar to the lexical entry for *hit*

default overriding is difference

extensional classes and concrete prototypes

Operationally, the two notions are more or less the same (Lascarides and Copestake 1999)
Prototypes

Some obvious problems

Grammar development

Is it possible to construct and maintain differential networks like this?

Types as generalization

A taxonomic approach to the lexicon encodes the fact that there are many more verbs than there are kinds of verbs

Multiple inheritance

Words and constructions can be related to each other along multiple orthogonal dimensions
Large scale grammar of English (Flickinger & Copstake 2000, Flickinger et al. 2000)

Implemented in the LKB

Organized around a large, detailed type hierarchy

Aimed at broad-coverage deep parsing and generation

The included lexicon (lexicon.tdl) lists 8,472 verb lexemes representing 336 types

Ten most frequent verb types account for 6,283 lexemes, and 135 verb types have only one member

v_np_le 1,723
v_np*_le 962
v_p-np_le 896
v_p_le 506
v_pp_e_le 494
v_-_le 463
v_np_noger_le 408
v_-_unacc_le 325
v_np-pp_e_le 322
v_pp*_dir_le 184
Inverse power-law distribution (Zipf’s Law)
Inverse power-law distribution (Zipf’s Law)
Scale invariance: Sublexicon of 800 randomly selected verbs (96 types)
Scale invariance: Sublexicon of 800 randomly selected verbs (96 types)
The ERG covers only a small part of the English vocabulary

Even for words that are listed, entries are incomplete (Baldwin, et al. 2004)

Suppose we constructed a lexicon with 100% coverage of the BNC . . . How many types would we need?
A verb frame is a bag of relations

\[ \text{persuade} \leadsto \langle \text{nsubjpass, advmod, xcomp} \rangle \\
\text{drop} \leadsto \langle \text{xsubj, dobj} \rangle \]
A **verb type** is a collection of frames that a verb occurs in

<table>
<thead>
<tr>
<th>verb</th>
<th>frame</th>
<th>count</th>
<th>verb</th>
<th>frame</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>persuade</td>
<td>xcomp</td>
<td>469</td>
<td>drop</td>
<td>nsubj dobj</td>
<td>594</td>
</tr>
<tr>
<td></td>
<td>xsubj xcomp</td>
<td>317</td>
<td></td>
<td>nsubj dobj prep</td>
<td>526</td>
</tr>
<tr>
<td></td>
<td>nsubj xcomp</td>
<td>316</td>
<td></td>
<td>nsubj prep</td>
<td>444</td>
</tr>
<tr>
<td></td>
<td>dobj</td>
<td>254</td>
<td></td>
<td>dobj</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>dobj xcomp</td>
<td>221</td>
<td></td>
<td>prep</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>dobj ccomp</td>
<td>144</td>
<td></td>
<td>dobj prep</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>nsubjpass xcomp</td>
<td>135</td>
<td></td>
<td>nsubj dobj</td>
<td>252</td>
</tr>
<tr>
<td></td>
<td>xsubj dobj</td>
<td>135</td>
<td></td>
<td>nsubj dobj advmod</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>nsubj dobj</td>
<td>126</td>
<td></td>
<td>nsubj advmod prep</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>nsubj dobj xcomp</td>
<td>112</td>
<td></td>
<td>nsubj prep prep</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Verb frames with the highest type frequency

<table>
<thead>
<tr>
<th>Structure</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>15,982</td>
</tr>
<tr>
<td>dobj</td>
<td>13,611</td>
</tr>
<tr>
<td>nsubj dobj</td>
<td>13,574</td>
</tr>
<tr>
<td>nsubj ccomp</td>
<td>11,347</td>
</tr>
<tr>
<td>prep</td>
<td>9,879</td>
</tr>
<tr>
<td>nsubj prep</td>
<td>7,878</td>
</tr>
<tr>
<td>dobj prep</td>
<td>6,987</td>
</tr>
<tr>
<td>nsubj dobj prep</td>
<td>6,873</td>
</tr>
<tr>
<td>nsubj xcomp</td>
<td>5,980</td>
</tr>
<tr>
<td>nsubj dobj advmod</td>
<td>5,843</td>
</tr>
</tbody>
</table>
Applying this method to the BNC, we get

- 92,612 distinct frames
- 67,423 verb lexemes
- 28,778 verb types

For each lexeme, drop frames that occur fewer than 10 times:

- 4,399 distinct frames
- 67,423 lexemes
- 2,554 lexical types

And if we also only consider lexemes that occur at least 500 times:

- 4,398 distinct frames
- 1,546 lexemes
- 1,545 lexical types
Verbs in the BNC do not appear to be organized into types

Is the lexicon structured at all?

Verb frames could be interpreted as binary features which define ‘natural’ classes of verbs

Or, verbs could be organized into differential network

What evidence is there for internal structure?
Spanning trees

A delegation network is a connected acyclic graph (spanning tree) joining all lexical entries.

Because lexical constraints are defaults, any network structure will work – but, not all are equivalent.
Spanning trees

Evaluate networks on the basis of shared information:

Measure the difference between joined lexical entries by Jaccard distance

\[ J_\delta(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|} \]

This captures the degree of default overriding between joined entries

A link between identical lexical entries would have a cost of 0

Find a minimum spanning tree – one with the smallest possible sum of edge weights (Kruskal 1956)
Spanning trees

The minimum spanning tree cost for BNC verbs is 597.00

Is that high or low?

Generate 100 uniform random (not necessarily minimum) spanning trees (Broder 1989, Aldous 1989)

Average sum of distances is 1227.69

Min is 1216.90 and max is 1239.14

Conclusion:

There aren’t many more verbs than there are types of verbs

Verbs also aren’t all unique

A differential network captures at least some of the structure in the verbal lexicon
Ginsberg and Sag (2000) present an analysis of a range of English interrogative constructions (and other related phenomena)

Detailed syntactic and semantic model based on HPSG and (more loosely) Situation Semantics

Constructions are organized into a multiple inheritance type hierarchy with a limited degree of default overriding

Location in the hierarchy specifies a constructions syntactic and semantic properties

### Declarative and interrogative constructions

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>decl_hd_su_cl</code></td>
<td><em>Kim smiled.</em></td>
</tr>
<tr>
<td><code>inv_decl_cl</code></td>
<td><em>doesn’t Kim like ___</em></td>
</tr>
<tr>
<td><code>decl_ns_cl</code></td>
<td><em>to smile</em></td>
</tr>
<tr>
<td><code>decl_frag_cl</code></td>
<td><em>Bagels.</em></td>
</tr>
<tr>
<td><code>pol_int_cl</code></td>
<td><em>Did Kim leave?</em></td>
</tr>
<tr>
<td><code>ns_wh_int_cl</code></td>
<td><em>What did Kim see?</em></td>
</tr>
<tr>
<td><code>su_wh_int_cl</code></td>
<td><em>Who left?</em></td>
</tr>
<tr>
<td><code>repr_int_cl</code></td>
<td><em>You’re leaving?</em></td>
</tr>
<tr>
<td><code>dir_is_int_cl</code></td>
<td><em>Kim saw Sandy?</em></td>
</tr>
<tr>
<td><code>slu_int_cl</code></td>
<td><em>Who?</em></td>
</tr>
</tbody>
</table>
### Other clause types

<table>
<thead>
<tr>
<th>Construction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv_excl_cl</td>
<td><em>Am I tired!</em></td>
</tr>
<tr>
<td>wh_excl_cl</td>
<td><em>how odd it is</em></td>
</tr>
<tr>
<td>ns_imp_cl</td>
<td><em>Be quiet!</em></td>
</tr>
<tr>
<td>top_cl</td>
<td><em>The bagels, I like.</em></td>
</tr>
<tr>
<td>factive_cl</td>
<td><em>that Kim left</em></td>
</tr>
<tr>
<td>root_cl</td>
<td><em>Kim left.</em></td>
</tr>
<tr>
<td>cp_cl</td>
<td><em>whether Kim left</em></td>
</tr>
</tbody>
</table>
Constructions

Non-clauses

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>fin_vp</td>
<td>went home</td>
</tr>
<tr>
<td>nf_hc_ph</td>
<td>going home</td>
</tr>
<tr>
<td>bare_nom_ph</td>
<td>old bagels</td>
</tr>
<tr>
<td>bare_adj_ph</td>
<td>very sad</td>
</tr>
<tr>
<td>nom_int_ph</td>
<td>who left</td>
</tr>
<tr>
<td>cq_np</td>
<td>Your name?</td>
</tr>
</tbody>
</table>
Constructions

Jaccard distance

![Diagram showing the Jaccard distance concepts related to CLAUSALITY and HEADEDNESS, with nodes like phrase, clause, core_cl, inter_cl, hd_ph, hd_only_ph, is_int_cl, repr_int_cl, decl_cl, sai_ph, and inv_decl_cl.]
Constructions
Constructions

- **Declarative**
  - `repr_int_cl`
  - `decl_hd_su_cl`
  - `decl_ns_cl`
  - `ns_imp_cl`
  - `inv_decl_cl`
  - `nf_hc_ph`
  - `ns_wh_int_cl`
  - `wh_excl_cl`

- **Interrogative**
  - `slu_int_cl`
  - `dir_is_int_cl`
  - `decl_frag_cl`
  - `factive_cl`
  - `cactive_cl`

- **Exclamative**
  - `cq_np`
  - `fin_vp`
  - `ger_cl`
  - `su_wh_int_cl`
  - `top_cl`

- **Non-clause**
  - `root_cl`
  - `nom_int_ph`
  - `bare_nom_ph`
  - `bare_adj_ph`
  - `cp_cl`

- **Other**
Constructions

The diversity is among constructions is lower than would be expected if HEADEDNESS and CLAUSALITY really were orthogonal dimensions.

A flat differential network captures most (all?) of the generalizations that G&S’s complex multiple inheritance hierarchy does.

Differential and hierarchical analyses aren’t mutually exclusive options (cf. traits).

Approaching the problem of organizing constructions quantitatively may reveal patterns that aren’t otherwise obvious.
Prospects

Differential networks are a viable alternative to taxonomic representations

How far can they be extended?

Richer datasets

Other lexicalist frameworks (Network Morphology, Word Grammar)

How can they be refined?

**Families** as a step towards types (Astudillo and Schilling 1993)

No reason to limit focus to spanning trees (Ackerman and Bonami)

Types, tokens, exemplars (Abbot-Smith and Tomasello 2006, Baayen *et al.* 2007)