October 23, 2003

Chapter 11.3–11.6

Feature Structures, Unification

Review

- Feature structures are sets of pairs of features and values.
- Values may be atomic symbols or feature structures themselves.
- The values of two or more features may be constrained to be (token) identical.
- In some systems, feature structures are typed.
- Types allow statements of feature appropriateness, and of constraints (on feature values) that apply to objects of the same class.

Review

- Feature structures are often formalized with and implemented as DAGs, although other formalizations are possible.
- The operation of unification checks whether two feature structures are compatible, and if so, combine the constraints from each into a single feature structure.
- Other operations on feature structures are conceivable (cf. $=_c$ constraints in LFG).

Outline of today's lecture

- Using features and unification to account for long distance dependencies
- Using features and unification for compositional semantics
- A unification algorithm
- Integrating a unifier into the Earley parser
- Further issues that arise in parsing with unification

Long distance dependencies: Questions

- Which book did you say that you thought Kim liked _ ?
- *Which book did you say that you thought Kim liked *The Wizard of Oz*?
- To which author has Kim written several fan letters _ ?
- Which author has Kim written several fan letters to _ ?
- *To which author has Kim written several fan letters to _ ?
- Which author has Kim written several fan letters?

Long distance dependencies: English focus movement

- Bagels, I like _ .
- Bagels, I know Kim likes _ .
- *Bagels, I know Kim likes lox.
- Of bagels, I know Kim is fond _ .
- Bagels, I know Kim is fond of.
- *Bagels, I know Kim is fond _ .
- *Of bagels, I know Kim is fond of _ .

Other long distance dependencies

- Relative clauses:

  This is the house that Jack built _ .

- *Tough* adjectives:

  This book is easy to read _ .

- *Tough* nouns:

  Those shoes are a challenge to walk in _ .

- Multiple LDDs:

  Violins this well crafted, these sonatas are easy to play _ on _ .

Three pieces of an LDD

- Bottom: recording the fact that something's missing.
- Middle: propagating the information that something's missing.
- Top: matching a filler to the gap, and sealing off the LDD.

Startgap: A traceless bottom

```
startgap-rule := unary-head-initial-sg &
[ SPR #spr,
  COMPS #comps,
  GAP <! #gap !>,
  ARGS < [ SPR #spr, COMPS < #gap . #comps > ]>].
```

Middle: Passing up GAP values in unary rules

```
unary-rule-pg := unary-rule &
[ GAP #gap,
  REL #rel,
  ARGS < [ GAP #gap, REL #rel ] > ].
```

Middle: Passing up GAP values in binary rules

```
binary-rule-pg := binary-rule &
[ GAP [ LIST #gfront, LAST #gtail ],
  ARGS < [ GAP [ LIST #gfront, LAST #gmiddle ] ],
         [ GAP [ LIST #gmiddle, LAST #gtail ] ]>].
```

Top: Pairing a filler with the gap

```
filler-head-rule := binary-head-final &
[ HEAD verb & #head,
  SPR <>,
  COMPS <>,
  GAP <! !>,
  ARGS < #gap & phrase & [ SPR <>, COMPS <> ],
         [ HEAD #head, SPR <>, COMPS <>,
           GAP <! #gap !> ] > ].
```

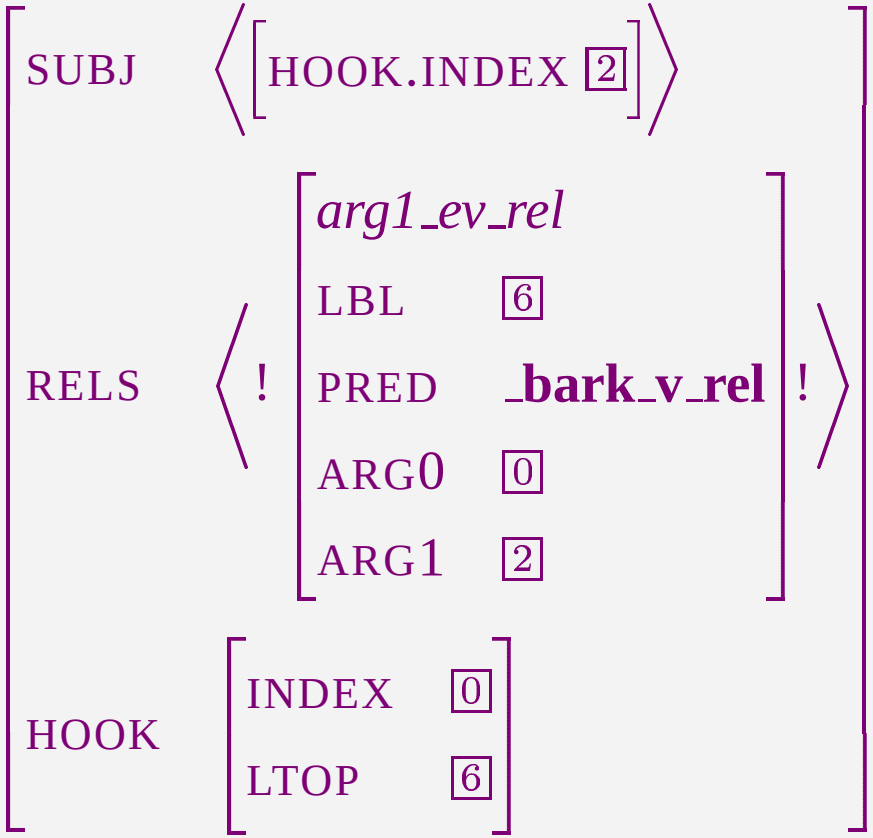
Compositional Semantics, using feature structures

- Minimal Recursion Semantics (MRS: Copestake et al 1999)
- Representations consist of a bag of elementary predicates (RELS), a bag of constraints on scopal relations between the predicates (HCONS), and a small set of features available for further composition (HOOK).
- Elementary predicates are all labeled by handles (which participate in the HCONS constraints).

An example MRS

$$
\begin{bmatrix}
\text{RELS} & \langle ! \begin{bmatrix} \textit{quant\_rel} \\ \text{LBL} & \boxed{1} \\ \text{PRED} & \textbf{\_def\_q\_rel} \\ \text{ARG0} & \boxed{2} \\ \text{RSTR} & \boxed{3}\ \textit{body}\ \boxed{4} \end{bmatrix}, \begin{bmatrix} \textit{noun\_rel} \\ \text{LBL} & \boxed{5} \\ \text{PRED} & \textbf{\_dog\_n\_rel} \\ \text{ARG0} & \boxed{2} \end{bmatrix}, \begin{bmatrix} \textit{arg1\_ev\_rel} \\ \text{LBL} & \boxed{6} \\ \text{PRED} & \textbf{\_bark\_v\_rel} \\ \text{ARG0} & \boxed{0} \\ \text{ARG1} & \boxed{2} \end{bmatrix} ! \rangle \\
\text{HCONS} & \langle ! \begin{bmatrix} \textit{qeq} \\ \text{HARG} & \boxed{3} \\ \text{LARG} & \boxed{5} \end{bmatrix} ! \rangle \\
\text{HOOK} & \begin{bmatrix} \text{INDEX} & \boxed{0} \\ \text{LTOP} & \boxed{6} \end{bmatrix}
\end{bmatrix}
$$

bark *says*


SUBJ
HOOK.INDEX 2
RELS
arg1_ev_rel
LBL 6
PRED _bark_v_rel
ARG0 0
ARG1 2
HOOK
INDEX 0
LTOP 6

The grammar rule says (in essence)

$$\begin{bmatrix}
\text{RELS} & \boxed{A} \oplus \boxed{B} \\
\text{HCONS} & \boxed{C} \oplus \boxed{D} \\
\text{HOOK} & \boxed{8} \\
\text{SUBJ} & \langle\,\rangle \\
\text{ARGS} & \left\langle \boxed{9}\begin{bmatrix}\text{RELS} & \boxed{A} \\ \text{HCONS} & \boxed{C}\end{bmatrix}, \begin{bmatrix}\text{SUBJ} & \langle \boxed{9} \rangle \\ \text{RELS} & \boxed{B} \\ \text{HCONS} & \boxed{C}\end{bmatrix} \right\rangle
\end{bmatrix}$$

Compositional semantics using feature structures

- Build up semantic representations alongside syntactic one.
- All semantic information is represented as constraints on signs.
- Lexical entries and rules both have semantic aspects to them.
- “The meaning of a phrase is a function of the meaning of its parts and the way they’re put together.”
- Phrases can also make substantive semantic contributions, beyond just tying things together.

Unification in Lisp: representing DAGs

```
(defstruct dag
  forward type arcs copy)

(defstruct arc
  feature value)
```

Unification in Lisp

```
(defun unify (dag1 dag2)
  (catch :fail (unify1 dag1 dag2)))
```

Unification in Lisp

```
(defun unify1 (dag1 dag2)
  (let* ((dag1 (deref dag1))
         (dag2 (deref dag2)))
    (unless (eq dag1 dag2)
      (let ((glb (glb (dag-type dag1) (dag-type dag2))))
        (when (null glb) (throw :fail nil))
        (setf (dag-forward dag1) dag2)
        (setf (dag-type dag2) glb)
        (loop ... [see next slide])
    dag2))
```

Unification in Lisp

```
(loop
    for arc1 in (dag-arcs dag1)
    for arc2 = (loop
                    for foo in (dag-arcs dag2)
                    when (eq (arc-feature foo)
              (arc-feature arc1))
                    return foo)
    when arc2 do
      (unify1 (arc-value arc1) (arc-value arc2))
    else do
      (setf (dag-arcs dag2) (cons arc1 (dag-arcs dag2)))))
```

Integrating unification into the Earley parser

- The only change is to COMPLETER, which needs to check whether the fs of the completed edge is compatible with the daughter it (apparently) matches in each incomplete edge.
- Why are there no changes to PREDICTOR or SCANNER?

A more radical approach to parsing with unification

- Replace category labels completely with feature structures (this is what's done in the LKB).
- Allows rules to UNDERSPECIFY the information that would have been in the category labels, and instead constraint only semantics, or only identify of category, or ...

A still more radical approach to parsing with unification

- Do away with CFG ‘backbone’ altogether.
- Take advantage of a recursive feature like ARGS (cf assignments 2 and 3).
- Parsing can be seen as successively resolving the ARGS values to fully specified feature structures.

Packed charts and unification

- Check for subsumption rather than equality.
- Leave ‘accumlator’ features (RELS, HCONS) out of the comparison.
- Unpacking becomes a bit more complicated.

Summary

- More examples of linguistic uses of feature structures
- Unification algorithm
- Issues pertaining to parsing with unification

Next time

- Review for midterm
- Put material covered so far into perspective
- Any requests?