*April 5, 2004*

*What we’re working with*

*More HPSG basics*

## *Overview*

- Components/workflow
- HPSG (signs, features/values, types)
- tdl syntax
- Feature geometry
- Valence features and grammar rules
- Overgeneration and undergeneration
- Apparent redundancies
- Word lists

## *Components*

- emacs – a text editor
- LKB – a grammar development environment
- HPSG (Head-driven Phrase Structure Grammar) – a theory of syntax
- tdl (Type Description Language) – a formalism/machine-readable language for HPSG
- Grammar Matrix – a grammar starter-kit, including grammar files written in tdl and LKB interface files written in lisp

## *Workflow*

- Develop an analysis (HPSG)
- Open the relevant file(s) in emacs
- Code the analysis in tdl, using types defined in the Matrix
- Save the files (emacs)
- Load/reload the grammar in the LKB, and check for errors
- In emacs, correct tdl syntax errors and analysis bugs (HPSG)
- Reload the grammar in the LKB
- ... rinse and repeat ...

## *Questions*

- What’s the difference between a parser and a grammar?
- Is the Grammar Matrix a parser or a grammar?
- Is emacs linguistic software?

## *What does the script file do?*

- Grammars can and often are be spread across several files.
- In addition to .tdl files, grammars can include lisp files specifying certain parameters to the LKB.
- The script file (in matrix/lkb/script) tells the LKB which files to load, and how to treat them when it loads them.
- Trying to load esperanto.tdl in the LKB will just result in an error.
- Forgetting to specify esperanto.tdl will mean that the LKB doesn’t load that file, missing anything you specify there.

## *HPSG: Language as a system of signs*

- The fundamental unit of language is a *sign*: a pairing of form and meaning (cf. de Saussure).
- Meaning is semantics and pragmatics.
- Form is syntax and orthography/phonology.
- Phrases and words (and lexemes) are both types of sign.
- Thus every node in the tree has form and meaning – semantics, orthography/phonology, and syntax are all built in parallel.

## *Features and values (1/2)*

- Feature/value pairs are a way to encode characteristics of objects.
- For a given feature, only some values are appropriate.
- E.g., crust type is a characteristic of pizzas.
  - Assume a feature CRUST
  - What are some appropriate values for CRUST?
  - What are some inappropriate values?

## *Features and values (2/2)*

- For a given object, only some features are appropriate:
  - What other features might be appropriate for pizzas?
  - What objects might a feature CRUST be inappropriate for?

## *Types*

- Types are a way of representing different kinds of objects.
- Types can bear three kinds of information:
  - Relation to other types (subtype, supertype, mutually incompatible type)
  - Feature appropriateness declarations
  - Constraints on particular feature values
- Atomic types (or ‘sorts’) don’t have any features at all.

## *Example from the Matrix (simplified)*

```
sign := avm &
  [ SYNSEM synsem,
    ARGS list,
    STEM list ].
```

(Note to self: Write this on the board)

## *tdl syntax*

- := – left is defined/constrained as on the right
- & – conjoins constraints
- [ ] – delimits feature structures
- . – end of type def, also used in path abbreviations
- , – separates feature-value pairs within a single fs
- ⟨ ⟩ – abbreviation for lists
- ⟨! !⟩ – abbreviation for diff lists
- #[a-z]+ – coreference tags for identity constraints

## *More types from the Matrix (simplified)*

```
lex-item := sign.
phrase := sign.

synsem := avm &
 [ LOCAL local,
   NON-LOCAL non-local ].

cat := avm &
 [ HEAD head,
   VAL valence ].
```

```
local := avm &
 [ CAT cat,
   CONT mrs,
   ARG-S list ].

valence :=avm &
 [ SUBJ list,
   SPR list,
   COMPS list,
   SPEC list ].
```

$$\begin{bmatrix} \textit{sign} & \\ \text{SYNSEM} & \begin{bmatrix} \textit{synsem} & \\ \text{LOCAL} & \begin{bmatrix} \textit{local} & \\ \text{CAT} & \begin{bmatrix} \textit{cat} & \\ \text{HEAD } \textit{head} & \\ \text{VAL} & \begin{bmatrix} \textit{valence} & \\ \text{SUBJ} & \textit{list(synsem)} \\ \text{SPR} & \textit{list(synsem)} \\ \text{COMPS} & \textit{list(synsem)} \\ \text{SPEC} & \textit{list(synsem)} \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \textit{mrs} \\ \text{ARG-S} & \textit{list(synsem)} \end{bmatrix} \\ \text{NON-LOCAL} & \textit{non-local} \end{bmatrix} \\ \text{ARGS} & \textit{list(sign)} \\ \text{STEM} & \textit{list} \end{bmatrix}$$

## *Observations about feature structures*

- Recursive: *synsems* within *synsems*, *signs* within *signs*.
- Need to specify a path from the outermost brackets to make it clear which instance of, say, CAT you mean.
- Complete parses aren’t actually trees, but big feature structures.
- How is constituent structure represented in these feature structures?
- The features don’t mean anything or do anything unless something else in the grammar interprets them.

## *Valence features and grammar rules*

- HPSG puts the bulk of the work on the lexicon.
- Words specify what their valence requirements are (SUBJ, COMPS, SPR).
- Phrase structure rules are not POS-specific.
- E.g., no VP $\rightarrow$ V NP NP.
- Rather, the rules put together a head with whatever dependents it says it's looking for.

## *Example: basic-head-subj-phrase (simplified)*

```
basic-head-subj-phrase := head-valence-phrase &
                          binary-headed-phrase &
  [ SYNSEM.LOCAL.CAT [ VAL [ SUBJ olist,
                             COMPS #comps,
                             SPR #spr ] ],
    HEAD-DTR.SYNSEM.LOCAL.CAT
            [ VAL [ SUBJ < #synsem >,
                    COMPS #comps,
                    SPR #spr ]],
    NON-HEAD-DTR.SYNSEM #synsem& canonical-synsem&
       [ LOCAL [ CAT [ VAL [ SUBJ olist,
                             COMPS olist,
                             SPR olist ]]]]].
```

## *Example: basic-head-subj-phrase (1/2)*

- HEAD-DTR has a non-empty SUBJ list.
- NON-HEAD-DTR's SYNSEM matches the thing on HEAD-DTR's SUBJ list.
- SYNSEM of mother gets lots of information from SYNSEM of HEAD-DTR (due to constraints inherited from supertypes)
- Other valence requirements copied up.
- Specializing this phrase involves linking HEAD-DTR and NON-HEAD-DTR to the two (cf *binary-headed-phrase*) members of the ARGS list.

## *Example: basic-head-subj-phrase (2/2)*

- Example: *Kim slept*
  - What's the SUBJ value of *slept*?
  - What's the COMPS value of *slept*?
  - What's the HEAD value of *slept*?
  - What's the SUBJ value of *Kim slept*?
  - What's the COMPS value of *Kim slept*?
  - What's the HEAD value of *Kim slept*?

## *head-final/head-initial (simplified)*

```
head-initial := binary-headed-phrase &
  [ HEAD-DTR #head,
    NON-HEAD-DTR #non-head,
    ARGS < #head, #non-head > ].

basic-head-final := binary-headed-phrase &
  [ HEAD-DTR #head,
    NON-HEAD-DTR #non-head,
    ARGS < #non-head, #head > ].
```

## *Overgeneration and undergeneration*

- Overgeneration:
  - Parsing ungrammatical sentences
  - Too many parses of grammatical sentences
  - Too many edges in the chart
- Undergeneration:
  - Not parsing grammatical sentences
- Grammar fragment

## *Apparent redundancies*

- lexical item designators and orthographies
- ARG-S and SPR/SUBJ/COMPS
- *head* types and *lex-item* types
- SPEC and SPR

## *Bring for next time*

- Some standardized words (let’s choose...)
- Update your matrix.tdl file from the web page
- Everything you brought & did last time

## *Overview*

- Components/workflow
- HPSG (signs, features/values, types)
- tdl syntax
- Feature geometry
- Valence features and grammar rules
- Overgeneration and undergeneration
- Apparent redundancies
- Word lists