An Introduction to Grammar Engineering using HPSG: DAY 4

Goals

1. Rework the grammar to make the lexeme – word distinction and add inflection.
2. Add a lexical rule for diathesis alternation (dative shift).

Background
Rework the grammar in order to make the distinction between lexemes and words and to add inflection in order to remove redundancy in the lexicon. Following the [Sag & Wasow, 1999] textbook, we will make every entry in the lexicon a subtype of lexeme, but require that the grammar rules operate on structures which are subtypes of word (as with the current grammar). This means that every lexical entry will have to be converted from lexeme to word via a rule. Some of these rules (plural etc) have morphological effect.
For a change, we provide you with a starting grammar that produces error messages when loaded and does not parse any sentences; half-way into the exercise we will arrive at a working version of the grammar. Obtain this fine grammar by issuing the command `cvs checkout grammar4`. This exercise involves a considerable amount of rearrangement of the grammar files. This is, unfortunately, something that grammar engineers end up doing quite often in real life. Take things step-by-step.

Exercises

1. Move the ARGS attribute from *phrase* to a higher type which includes both words and phrases.
2. Add the top level lexeme – word distinction. We do this by distinguishing between two types, *word* and *lexeme* which inherit from a new type *lex-item*. The feature ORTH is introduced on *lex-item*. The type *word* will be a supertype of all the lexeme-to-word rules. The structures you need are given below.

```
lex-item := syn-struc &
[ ORTH string ].

lexeme := lex-item.

word := lex-item &
[ HEAD #1,
  SPR #2,
  COMPS #3,
  ARGS < lexeme &
         [ HEAD #1,
           SPR #2,
           COMPS #3 ]
         > ].
```

3. The old types *verb-word* et al. had constraints which should belong to lexemes under our new view of the world. Add subtypes of *lexeme* for *verb-lxm*, *noun-lxm* etc., which use the constraints from the old *verb-word*, *noun-word* etc. Make *det-lxm* and *prep-lxm* both inherit from a type *const-lexeme*, defined as follows:

```
const-lexeme := lexeme.
```

4. Now modify your '`lexicon.tdl`' file:
   (a) Remove all entries which are not morphologically equivalent to base forms (e.g. *dogs*). If your file has *gave* but not *give*, replace the entry for *gave* with one for *give*.
   (b) Change the types on the remaining entries so they are all subtypes of lexeme appropriate for the entry. Your file should now consist of base forms with just base orthography, a single type, and valence-specific COMPS values on verbs.

5. At this point, loading your grammar should no longer result in error messages printed to the 'LKB Top' window, and you should again be able to parse sentences. If there are still load-time errors, make sure that all required types (see above) are defined (and that their names are not mispeled). Test the functionality of the resulting grammar, using the batch parse machinery on the '`test.items`', '`agr.items`', and '`mod.items`' test suites. If there are surprises, make the necessary corrections.

   We have given you a new file '`inflr.tdl`' which defines the actual inflectional rules. Please do not worry too much about the lines beginning with %, these are instructions to the orthographemic component, relating the application of each rule to a specific variation in spelling. Take a look at '`inflr.tdl`' in `emacs` and convince yourself that all inflectional rules map arguments of type *lexeme* to signs of type *word* (which can then act as arguments to syntactic rules).

6. Study the parse tree for the sentences 'The dog barks near the cat' at great length, and ovserve how the different types of lexemes are mapped to words by the inflectional machinery.

7. Next, we will introduce another type of lexical rules, viz. a lexeme-to-lexeme lexical rule, to capture another generalization and further eliminate redundancy in '`lexicon.tdl`. In English, most ditransitive verbs of two NP complements (*give*, *send*, *sell* et al.) can undergo a lexical process known as *dative shift*, resulting in a variant of the verb where the second NP argument has been promoted to the first argument position, and the original first NP argument turns into a second PP argument, headed by the preposition *to*. For example, dative shift turns *Kim gave Sandy a book* into *Kim gave a book to Sandy*. To account for this alternation in argument structure, we will add a new lexical rule, turning one verbal lexeme into another verbal lexeme. The lexeme-to-lexeme type of lexical rules is, by nature, not associated with orthographemic variation.

   Open the empty file '`lrules.tdl`' and add a new type which has the following structure:

```
dative-shift_lrule := verb-lxm &
[ HEAD ...,
  SPR ...,
  COMPS ...,
  ARGS < verb-lxm & [ ... ] > ].
```

8. Fill in any necessary constraints for each attribute, so that the rule takes as its single argument a ditransitive verb with two NP complements, and produces a ditransitive verb with an NP complement and a PP complement. In our current grammar, we have no mechanism to select for a specific preposition, as would be required in this case. If you got to this point quickly, experiment with a HEAD feature PFORM on prepositions to make the grammar more precise.

9. Remove your hand-built lexical entry for the NP – PP version of *gave* from the file '`lexicon.tdl`' since we now have a productive lexical rule which generates this entry for you.

10. Realize that we did not have to encourage you to test the revised grammar thoroughly; at this point, you are using the 'Parse Input' and 'Batch Parse' facilities of the LKB regularly to monitor the effect of changes to the grammar.