An Introduction to Grammar Engineering using HPSG: DAY 3

Goals:

1. Learn more about typed feature structures and unification.
2. Finish treatment of modification.
3. Improve analysis of agreement
4. Capture generalizations in lexicon and rules

Exercises:

1. (Continued from Day 2) Extend the grammar to provide an analysis of modification, admitting sentences like *The dog barks near the cat.* We introduce a new head feature MOD and a new syntactic rule for modifiers, and we make use of the notion of underspecification. If you want to move on—leaving completion of the previous exercise(s) to when you get back home—obtain a fresh starting grammar by issuing the command `cvs checkout grammar3`.

2. The constraints we added yesterday to enforce subject-verb and determiner-noun agreement get the facts pretty much right, but from a grammar engineering point of view leave a lot to be desired.

   - Since the feature AGR is introduced on the type *pos*, all kinds of words will have an agreement feature. However, in English only determiners and nouns have agreement information. Unused features unnecessarily increase the size of the grammar and can make errors more difficult to track down. Modify your grammar so that the feature AGR only appears on *det* and *noun*, and not on *verb* or *prep*. To do this, you will need to add a new type, *agr-pos*, which is a subtype of *pos* and a supertype of *det* and *noun*. Be sure to check your work using the batch parse mechanism.
   - The intuition behind determiner-noun agreement is that the AGR value of the noun must be the same as that of its specifier. In our grammar, though, the AGR value of the noun and the AGR value of the specifier are stipulated separately. Use re-entrancies to eliminate this redundancy and capture the generalization underlying agreement. Again, verify your changes by parsing your test sentences.

     In case the syntax of re-entrancies in TDL is still confusing, here's an example:

     ```
     x := y &
       [ F #1 & z,
         G #1 ].
     ```

     This definition means that the value of the feature F is *z*, and the value of the feature G is the same as that of the feature F.

3. Those of you who are familiar with HPSG may be wondering why there is no Head Feature Principle in the grammar. Even if you aren't, you may have noticed that in each rule the HEAD value of the whole phrase is always the same as the HEAD value of one of the ARGs. The argument which contributes the HEAD value to the whole phrase is known as the *head* of the phrase. For some kinds of phrases, the head daughter is the first daughter, and for some it's the last daughter. Rearrange the hierarchy of rules to capture this distinction between head-initial phrases and head-final phrases.

   - In `types.tdl`, add three new types:

     ```
     head-initial := phrase &
      [ HEAD #head,
        ARGS < [ HEAD #head ], ...> ].
     head-final := phrase &
      [ HEAD #head,
        ARGS < syn-struc, [ HEAD #head ] > ].
     root-hf := root & head-final.
     ```

Note that our definition of *head-final* makes the simplifying assumption that all head-final phrases are binary, i.e. have exactly two daughters (which is true for our current grammars). Also, we will have more to say about the types *root* and *root-hf* later in the course.

- Modify the rules in `rules.tdl` to inherit from these new types. For example, the *head-complement-rule-0* should look like:

```
head-complement-rule-0 := head-initial &
[ SPR #spr,
  COMPS < >,
  ARGS < word &
         [ SPR #spr,
           COMPS < > ] >  ].
```

We call this rule head-initial, even though it has only one daughter, since the other head complement rules are also head-initial. Head-final rules, like the *head-specifier-rule*, should inherit from *head-final*. The feature HEAD should not need mentioning in `rules.tdl` at all, except for one occurence in the *head-modifier-rule* perhaps.

4. Using the same strategy, find and eliminate more redundant specifications in the grammar. Improve the organization of the type hierarchy to make it easier to add new words that are similar to words already in the lexicon. As a place to start, take a look at the lexical entries for nouns, and note that the same information is stated again and again in each entry. State those generalizations as constraints on a new type, *noun-word*, which every noun lexical entry inherits from. As you work, use the batch parse facility to make sure none of your modifications have damaged the coverage of the grammar.