Grammar Engineering — ESSLLI 2016 (Exercise 4)

High-Level Goals

- Implement lexical rules for inflection and derivation
- Simplify the lexicon and increase expressivity using these lexical rules

1 Obtaining the Starting Grammar

- Bring up our development environment by opening a terminal ('`Control-Alt-T`') and running the '`lkb` command from the shell command line.

  We will start with a new version of the grammar for today's exercise, so get a copy of the starting grammar by typing at the shell prompt:

  ```
  svn co $grammar4
  ```

- This grammar is a somewhat enriched version of the solution grammar for the previous exercise. The lexicon no longer contains only fully inflected words, but rather uninflected lexemes which inherit from lexical types that express what is common to each of the instances of the type. For example, the type *noun-lxm-intransitive* unsurprisingly includes the constraint [`HEAD` *noun*] which was previously declared separately on each noun entry in the lexicon. This type also constrains the `SPR` and `COMPS` values as expected, and includes the reentrancy for agreement that you added separately to each noun entry in the previous exercise.
- We have added to the grammar a file called `irules.tdl` that defines inflectional lexical rules to build expressions of type 'word' out of these lexemes. This grammar also has an additional subtype of *expression* called *syn-struc* which enables us to distinguish the types *word* and *phrase* (which can appear as daughters of phrases) from *lexeme* which cannot, forcing them to undergo an inflectional rule in order to be available to construct phrases.
- The grammar also makes use of a few additional descriptive devices, all defined as types and feature structures. We have added a type *bool* ("boolean") whose two subtypes are simply '+' and '–', so that we can employ features like OPT (for 'optional') which are given a positive or negative value. You will also see two examples of a somewhat fancy concept called a *parameterized list*, which allows us to require that every member of such a list is compatible with some particular constraint; for example, it is convenient to have a notion of a (possibly empty) list all of whose members are consistent with the constraint [OPT +] meaning that they are optional.
- Notice that the `ARGS` attribute has moved up from *phrase* to *expression* so that we can also implement lexical rules as typed feature structures, for both lexeme-to-lexeme and lexeme-to-word rules. We will also want a second partitioning of our three subtypes of expressions, one which groups *word* and *lexeme* together, as subtypes which inherit from a new type *lex-item*, another subtype of *expression*. The feature `ORTH` is now introduced on *lex-item*, since we will not surprisingly care about orthography for both lexemes and words. The type *word* will then be a supertype of all the lexeme-to-word rules. (This may seem a little surprising at first, having these new unary rules be subtypes of *word*, but have patience.).

2 Inflectional rules

- As noted, we have given you a new file `irules.tdl` which defines the actual inflectional rules. Take a tour of these rules, but don't worry too much about the lines beginning with `%suffix` since these are instructions to the orthographemic component, relating the application of each rule to a specific variation in spelling. Looking at the file, 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). Study carefully the parse tree for the sentence *The dog barks near the cat*, and observe how the different types of lexemes are mapped to words by the inflectional machinery.
- Now add one missing inflectional rule to this `irules.tdl` file, for verb words such as *chase* and *bark* which require a specifier whose `agr` value is *non-3sing*. Note that this rule does not need to change the orthography of the lexemes that it applies to. Test your improvement by doing a batch parse with the file `all.items`.

3 Derivational Lexical Rule: Dative

- Next, we will define a second type of lexical rule, namely a derivational rule that takes a lexeme as inputs and produces another lexeme, to further eliminate redundancy in the file "lexicon.tdl". In English, most ditransitive verbs with two NP complements (e.g. *give*, *send*, *sell*) 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 captures the alternation in the two sentences *Kim gave Sandy a book* and *Kim gave a book to Sandy*. To account for this alternation in argument structure, we will add a new lexical rule, deriving one verbal lexeme from another.

  Open the file `lrules.tdl` (for our derivational lexical rules) and add a new rule which has the following structure:

```
dative-shift-lrule := some-subtype-of-verb-lxm &
[ ORTH #orth,
  HEAD ...,
  ARGS < another-subtype-of-verb-lxm & [ ORTH #orth, ... ] > ].
```

- Choose suitable input and output types, and fill in any necessary constraints for each attribute, so that the rule takes as its single argument (its input) a ditransitive verb with two NP complements, and produces as its output a ditransitive verb with an NP complement and a PP complement. Remember that in our implementation of lexical rules, the expression which the textbook labals as INPUT is the single element of our `args` list, while the type which the textbook labels as the OUTPUT type is what appears immediately after the `:=` definition symbol in our rules.

- Remove the hand-built lexical entry for the NP – PP version of `give` from the file `lexicon.tdl` since we now have a productive lexical rule which generates this entry for you. Your grammar should now account nicely for the dative alternation. Test appropriately, and, as always, consider adding test items to `all.items`.

4 Another Derivational Rule: Agentive Nominalization

- While the dative rule has no morphological effect (no spelling change for the two variants), the next lexical rule does, and defining it will make use of the LKB orthographemic component, i.e. the facility to associate a change in orthography with a lexical rule. You should be able to work from the examples provided in `'irules.tdl'` to see how this spell-changing machinery is expressed in our lexical rules.

- Add a second derivational lexical rule to the file "lrules.tdl", this one deriving agentive nouns from verbs, to provide an analysis for sentences like *The barkers chased those cats* and *The chasers barked.*, without adding any additional noun or verb lexemes to `lexicon.tdl`. Adapt the orthography-changing machinery (taking inspiration from the inflectional rules), to add the "-er" suffix to the input orthography of this new rule. The rule will need to include this line for the "-er" suffix, which says that a word ending in 'e' just adds an 'r', while a word not ending in 'e' adds an 'er':
  `%suffix (e er) (!e !eer)`

  Add relevant test items (both grammatical and ungrammatical) to the file `'all.items'`, and check your analysis for both overgeneration and undergeneration.

5 Extend Agentive Nominalization: Optional Complements

- Extend your analysis of agentive nouns to allow for an optional complement PP marked with *of*, as in *The chasers of the cats barked.* You may want to look at the lexeme type for the verb *climb* in `lexicon.tdl` to see how we accommodate optional complements, and you will likely also find the type *noun-lxm-transitive* useful. Note that your improved analysis might still employ only one lexical rule to derive agentive nouns, but you might well find it convenient to define more than one such lexical rule. Be sure that your improved analysis still accepts *The chasers barked*, but not **The chasers to the cats barked.* Add more test examples to the file `'all.items'` to test your analysis.