Formal Syntax & Grammar Engineering (Exercise 2)

High-Level Goals

- Become familiar with the LKB grammar development system.
- Learn to extend the grammar by adding lexical entries.
- Implement a unification-based account of agreement constraints.

1 Starting the LKB — Analyzing Sentences

- We will be using a central server (borrowed from GSLT) to run the LKB grammar engineering environment. At the (X11) shell prompt, type:

  ```
  ssh mozart.gslt.hum.gu.se -l fsem-42
  ```

  and login with the appropriate *fsem* password (i.e. using the course account that you were assigned).

- From the shell prompt (on '`mozart`'), obtain a copy of our starting grammar by typing:

  ```
  cvs checkout grammar1
  ```

- Start emacs(1) and the LKB: at the shell prompt, execute:

  ```
  lkbemacs
  ```

- Load the grammar by selecting Load | Complete grammar in the window called Lkb Top, then double-clicking on the directory '`grammar1`' and on the file '`script`'. Reassuring messages will appear in the Lkb Top window, and a window will pop up showing you the type hierarchy for this small grammar.
- With the mouse in the Lkb Top window, select Parse | Parse input... from the menu.
- Type in the sentence *the cat chased the dog*, thereby replacing the existing contents of the new window that pops up.
- Click on the button OK. The system will parse the sentence and pop up a window containing a little parse tree for the single analysis of this sentence. Click on the parse tree to get a menu which allows you to enlarge it and look at the nodes in more detail.

2 Try the Simple Batch Parsing Mechanism

- In the Lkb Top window, select the menu item Parse | Batch parse... which will pop up a window asking you for an input file to be processed.
- Click on the file '`test.items`' in your grammar directory, then hit the button OK. This will pop up a new window asking you for the name of the output file where the results of the batch run will be stored; choose '`test.results`' for the output file and confirm that, indeed, you want this file to be overwritten.
- The system will print the message *Parsing test file* in the `*common-lisp*` buffer in emacs(1) when it starts, and will print the message *Finished test file* when it is done.
- Open the file '`test.results`' in emacs and inspect the parsing results.

3 Add a Lexical Entry for Another Animal Noun

- In emacs, open the file '`lexicon.tdl`' for editing.
- Copy the five lines that define the lexical entry for *cat* and modify your copy to make the value of `ORTH` appropriate for another animal; also, assign a new identifier (the name preceding '`:=`') to your entry.
- Save the changed version of the file.
- Reload the grammar and test the effect of your addition. In the Lkb Top window, execute Load | Reload grammar. Study the messages printed to the Lkb Top window; in case there are errors, correct your changes to '`lexicon.tdl`' and reload. Next, parse the sentence *the cat chased the animal* (substituting the name of your animal, of course).
- Add this sentence to the '`test.items`' file and rerun the batch check.

4 Poke Around the Grammar a Little

- Investigate the grammar in order to get an intuitive idea of how it works; we will discuss more formal details later. In particular, look at the following sentences and try and decide why they do or do not parse:

  the cat barks
  the cat chased
  cat barks
  the cat bark
  bark

- Note that the Parse | Show parse chart menu entry can give you an idea of which constituents were built, even when an input is not recognized by our grammar. Just like in the (enlarged) tree view, entries in the parse chart are mouse sensitive and allow inspection of the feature structure associated with each constituent. Notice that the grammar is parsing some sentences incorrectly (i.e. overgenerates) and failing to parse some sentences that should parse (i.e. undergenerates).

5 Adding a More Interesting Lexical Entry

- The rule that is needed for ditransitives (i.e. verbs that take two objects to their right) is in the grammar, but there are no lexical entries that utilize it. Add an entry for *gave* which takes two noun phrase complements (i.e. what is needed to parse, say, *that dog gave the cat the animal*).
- Copy the entry for *chased* in '`lexicon.tdl`'. Replace the orthography value as before and assign a new lexical identifier to this entry (e.g. '`gave`').
- Add an extra element to the `COMPS` list, which will be a duplicate of the one that is already there. Note that lists are delimited by angle brackts ('`<`' and '`>`') and the elements on lists are separated by commas.
- Test by parsing *that dog gave the cat the animal*. Also test for overgeneration by confirming that you cannot parse *that dog gave the cat*. Add an appropriate set of test sentences to '`test.items`'.

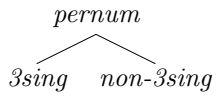
6 Introduce Prepositional Phrases and PP Complements

- In order to parse *that dog gave the cat to the animal*, we will have to extend our grammar futher.
- Add the type *prep* as a new subtype of the type *pos* to the file '`types.tdl`', by copying the type description for *noun* (and replacing *noun* with *prep*).
- Add a lexical entry for the preposition *to*. This should be similar to the entry for *chased* in that *to* will take a single noun phrase complement, but the value for `HEAD` should be *prep* and the value of `SPR` should be the empty list (i.e. '`<>`').

- Add a second lexical entry for *gave*. You can copy your existing entry but you will need to use a different identifier (i.e. the thing to the left of the '`:=`' operator), for example '`gave_np_pp`. You also need to change the second element in `COMPS` to make this entry require a PP (we will not bother about making sure it is a PP headed by *to* yet).
- Add several test items, both grammatical and ungrammatical, to your '`test.items`' file, which will allow you to check the correctness of your additions to the grammar.
- Run the batch parsing utility again, and examine the results.
- Celebrate as appropriate.

7 Subject – Verb Agreement

- Extend the grammar to capture subject – verb agreement, admitting e.g. *the dog barks* but not **the dogs barks*. We will introduce constraints on the `SPR` attribute of lexical entries requiring that person and number properties match between head and specifier. Rather than using separate features for number and person, we will use types that combine both properties, allowing a more direct encoding of English inflectional morphology.
- Add this small type hierarchy to the file '`types.tdl`', making *pernum* a subtype of *feat-struc*:



- Also in '`types.tdl`', add the feature `AGR` to the type *pos*, with its value constrained to be of the new type *pernum* that you just added.
- In the lexicon file, add the appropriate constraint to each verb by restricting the `AGR` value inside of its `SPR`. Also in the lexicon, add the correct `AGR` value to each noun.
- Save your changes, then reload the grammar, apply the batch test with the file '`agr.items`', examine your results, and make any necessary corrections (but ignoring determiner – noun agreement, for now).

8 Determiner – Noun Agreement

- Extend your analysis to cope with determiner – noun agreement, admitting e.g. *these dogs bark* but not *these dog barks*.
- In lexicon again, modify each noun's lexical entry by adding the appropriate constraint on the `AGR` value of its `SPR`. Also add the correct `AGR` value to each determiner.
- Check your revised grammar again using the file '`agr.items`', and make any necessary corrections.
- Add some additional test examples to this file with varied combinations of mismatch in agreement among determiners, nouns, and verbs. Then run the batch test and examine the results.

9 Phrase Structure Recursion

- As the LKB (and similar parsing systems) assume that each rule has a fixed number of daughters, we ended up with two instances of the head – complement schema: one binary-branching, picking up a single complement, the other ternary, picking up two complements (plus, strictly speaking, the third instance, which is unary and promotes intransitive lexical heads into phrases).
- To condense the binary and ternary instances of the head – complement schema into just one, investigate a binary-branching VP analysis. For a ditransitive head (e.g. *gave*), there will need to be two consecutive applications of the head – complement rule, each picking up one complement at a time and reducing the `COMPS` list appropriately. In the unlikely event that you get this to work, use the batch parsing machinery to make sure that there is no spurious ambiguity.

Submit your results in email to Stephan and Lilja by 18:00 h on Thursday, November 4.