Constraint Grammar Work
L445 / L545
Spring 2017

Main Documentation

Other than the Karlsson (1990) paper to introduce the constraint grammar (CG) framework, we'll rely on two documents (though, there are many others):

1. Tutorial slides: `http://visl.sdu.dk/~eckhard/powerpoint/CG3_Nodalida_dis.pdf`
2. How-to handout: `http://beta.visl.sdu.dk/cg3_howto.pdf`

(Pre)processing Commands

Let's start with some plain text file (e.g., from Project Gutenberg[1])

1. Run TreeTagger[2] on this data, but in a way that generates more than one possible tag per word:
   - I did this by copying `cmd/tree-tagger-english` and then replacing (line)

     `OPTIONS="-token -lemma -sgml"`

     with:

     `OPTIONS="-token -lemma -sgml -threshold 0.1"`
     - Note: you may want to try different thresholds
   - The actual command will be:

```
$ cmd/tree-tagger-MODIFIED FILE.txt > FILE.tts
```

2. Post-process the script into CG format. I have provided `tt2visl.py` for this purpose, but (caveat!) you may find you want to modify it at some point:

```
$ cat FILE.tts | python tt2visl.py > FILE.cg
```

3. Once you have a constraint grammar file (see next section), the running of `vislcg3` is as follows:

```
$ cat FILE.cg | vislcg3 --grammar GRAMMAR.gram > FILE.out
```

[1] `http://www.gutenberg.org`
[2] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

Grammar File

There are five basic sections to a constraint grammar file. Note that the order of rules matters (for CORRECTIONS, MAPPINGS, and CONSTRAINTS).

1. DELIMITERS

   ⇒ Define sentence boundaries

   ```
   DELIMITERS = "<.>" "<!>" "<?>" "<\;>" ; # sentence window
   ```

2. SETS

   ⇒ Define sets of tags, generally using the `LIST` or `SET` commands

   ```
   LIST ADJ = JJ JJR JJS;
   ```

   Define the category ADJ as being any of the JJ. tags*

3. CORRECTIONS

   ⇒ Define changes to tags (e.g., correct faulty input), generally using the `MAP` or `SUBSTITUTE` commands

   ```
   SUBSTITUTE (IN) (CS) TARGET ("that");
   ```

   Define a new POS tag, CS, for certain IN instances

4. MAPPINGS

   ⇒ Define mappings which add tags to readings (e.g., add functional labels on top of POS tags)

   - `ADD`: add readings (often functional labels, i.e., ones starting with `@`)
   - `MAP`: same as `ADD`, but cannot apply if there is already one or more `@` tags

   ```
   ADD (@SUBJ>) TARGET (NN)
       IF (1 (VBD));
   ```

   Add the `@SUBJ` functional label to NNs preceding VBD (past tense verb) tags

5. CONSTRAINTS

   ⇒ Define rules to disambiguate tags

   - `REMOVE`: remove a tag from a word's set
   - `SELECT`: select the one best tag from a word's set
   - Note that rules are rerun in order until no further rules can be run (see p. 3 of manual)

   ```
   SELECT ADJ IF (-2 (JJ)) ;
   ```

   Note: ADJ is a class (defined by `LIST`), whereas JJ Is a POS tag and thus has to be put into parentheses