Symbolic AI Coursework - Grammars, Parsing and Logic

March 2019

1 Motivation

In this exercise we will explore the connection between Natural Language, Grammars, Parsing and Knowledge Bases.

2 Setting

3 Pedagogical Reasoning or Why do I need to do this?

- There is a beautiful and powerful connection between language and logic. As part of this exercise we will expose you to this connection. We will see how two types of grammar (and syntax in general) can be used as the vehicle to go from sequence of symbols to representations of meaning (logical forms).
- In order to better understand the principles we will focus on simpler examples. However, the techniques that you will see here transcend the academic exercise: all of them can be applied to develop real-world AI systems.
- The goal of the exercise is to consolidate and expand on the material which we saw during the classes. In order to complete the lab you will need to do some research and learn new concepts. Reference recommendations are pointed. This pedagogical design decision makes the exercise more challenging, engaging and richer.

4 Rules of the Game

This task should be done individually.

5 Marking

The marking scheme is flexible as some of the questions support multiple answers. The goal is to approach the exercise from a real-life perspective (not as an overly prescribed task). The distribution of marks is uniform across each exercise.

6 Submission

You will submit this report together with the previous lab report in different PDF files.

7 What do I need to do?

- Please organise your answers to the questions below in a single report (pdf).
- Please care about aesthetics and interpretability. Please make sure that your answers and diagrams are properly formatted. Latex is recommended for formatting your output.

8 Exercise 1 (Building a CFG)

Assume we have the following corpus of individual sentences:

```
Steel is an alloy.
Steel contains carbon.
```

1. Read the first section on Context Free Grammars (CFGs) here[1] .
2. Write down a Context Free Grammar (CFG) which is able to recognise these sentences. Start by listing the *POS tags* and the *phrasal nodes* which you will use. Then write down the grammar.

Tip: you can use a third-party constituency parser in case you are not secure around the tag sets ([2]). Use the Penn Treebank tagset for the task.

9 Exercise 2 (Building a CFG in Prolog (using Difference Lists))

Now that you have your grammar specified, let's build a Prolog parser to recognise and generate sentences using this grammar.

[1] `http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse28`
[2] `http://nlp.stanford.edu:8080/parser/index.jsp`

1. Read the last two sections here[3].
2. Translate the CFG grammar from the previous exercise into Prolog using difference lists.
3. Check if the grammar is able to recognise the four sentences.
4. Generate all the sentences which can be recognised by this grammar.
5. Include your program as a properly referenced appendix into your report.

10 Exercise 3 (DFGs)

Prolog provides a convenient way to represent CFGs, using Definite Clause Grammars (DFGs).

1. Read the following article[4] to learn about DFGs.
2. Rephrase the previous grammar into a DFG.
3. Check if you can recognise and generate the same sentences.
4. Include your program as a properly referenced appendix into your report.

11 Exercise 4 (CCGs)

Now we will look into an alternative way to encode grammars using Combinatory Categorical Grammars (CCGs). Given the following sets of natural language statements and questions:

```
Statements:

Steel is an alloy.
Steel contains carbon.

Questions:

Does Ferrite have high hardness?
Which material has the lowest tensile strength?
```

1. Recap CCGs using the slides of our lecture and this paper[5] on Semantic Parsing (until section 2.2 included).

[3] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse28
[4] http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse29
[5] https://homes.cs.washington.edu/~lsz/papers/zc-uai05.pdf

2. Create a CCG Grammar which recognises these sentences. Focus on the syntactic dimension of the grammar (do not include the lambda calculus features).

3. Draw the derivation trees for each sentence.

Tip: you can draw CCGs using this latex package: `https://github.com/bozsahin/ccg-latex`. Feel free to use other ways to draw them.

12 Exercise 5 (CCG Syntactic Parser)

NLTK is a lovely Python library for playing with natural language. We will use it to implement a syntactic parser for the grammar that you designed in the previous exercise.

1. Read the first section ('No semantics') here[6] to understand the basics. You can complement with this reference[7] to see how to model more sophisticated structures.

2. Implement a syntactic parser in NLTK for the grammar that you designed in the previous exercise.

3. Print the derivation tree for the four sentences.

4. Include your program as a properly referenced appendix into your report.

13 Exercise 6 (CCG Semantic Parser)

Now we will proceed to the next step and add a semantic layer to our grammar.

1. Continue reading the reference[8] to understand how to add semantic features into your sentences.

2. Add the semantic features into your grammar. Use the λ-calculus notation as in the examples.

3. Print the derivation tree for the sentences in the corpus using the semantic features.

14 Exercise 7 (Question Answering)

Now we will connect the questions to a knowledge base. Consider the following KB of facts:

[6] `https://github.com/nltk/nltk/blob/develop/nltk/test/ccg_semantics.doctest`
[7] `http://www.nltk.org/howto/ccg.html`
[8] `https://github.com/nltk/nltk/blob/develop/nltk/test/ccg_semantics.doctest`

```
Ferrite has 0.1 tensile strength.
Ferrite has low hardness.

Perlite has 0.3 tensile strength.
Perlite has medium hardness.

Austenite has 0.4 tensile strength.
Austenite has medium hardness.

Cementite has 0.7 tensile strength.
Cementite has high hardness.
```

and the questions: Does Ferrite have high hardness? Which material has the lowest tensile strength?

1. Manually convert the facts into a NLTK-style *lambda*-calculus logical form.
2. For the two questions, adapt the semantic features of your previous grammar to match the predicate form in the KB.
3. Run the parser and print the parse trees for the two questions.
4. How would you use the parser output to compute an answer for the query over the KB?