NLP Homework 3

Writing a CFG (Context-Free Grammar)

For this homework assignment, you will be developing a CFG for a small subset of English. There will be just one part, writing grammar rules, with no option for programming.

The grammar rules will be written in a file that can be read by the NLTK. The first part of the grammar file contains the structure rules for the phrases and the second part contains the lexical rules that assign non-terminal symbols for the words in the vocabulary of this limited language. These non-terminal symbols can be thought of as POS tags, but are not required to be.

You will have a set of sentences to try to parse, given in a file called sentences.txt. There will be a second file of sentences, challenge_sentences.txt, from which you will choose two more sentences to parse.

For the homework, download the CFGdevelopment.zip file and extract the directory that has the following structure:

```
CFGdevelopment/
        camelot_grammar.cfg
        challenge_sentences.txt
        sentences.txt
        development.py
```

The file development.py has a small bit of python that reads sentences from the file sentences.txt, reads the grammar from camelot_grammar.cfg, produces a recursive descent parser and parses the sentences, if possible. When you choose two challenge sentences, you can copy them to the end of the file sentences.txt.

You will add grammar rules to the file camelot_grammar.cfg to parse the sentences. The NLTK program that loads grammars from .cfg files will ignore blank lines and comment lines starting with #.

More details on grammars

More formally, a probabilistic context free grammar consists of:

- A set of non-terminal symbols
- A set of terminal symbols (which we can call the vocabulary)
- A set of rewrite or derivation rules
- A start symbol

For natural language CFGs, we think of the start symbol as indicating "sentence" (in this case it will be START), and the terminal symbols as the words.

A derivation rule gives one way to rewrite a non-terminal symbol into a sequence of non-terminal symbols and terminal symbols. For example, S -> NP VP says that an S (perhaps indicating a declarative sentence) can be rewritten as an NP (noun phrase) followed by a VP (verb phrase). For the system we've provided you with, the rules need to be written one per line in a plain text file with the following syntax:

non-terminal symbol -> option1 | option2. . .

Each of the right-hand-side options can be a sequence of non-terminal symbols or word/terminal symbol.s   Any word symbol must appear in quotes. So, for example, you could have a rule in the vocabulary file that always requires the words Holy and Grail to occur in sequence as a proper noun:

NNP -> ‘Holy Grail’

Words/terminal symbols can occur both on the right hand sides of rules in S1 and in the lexical rules, e.g. the word “was” can occur as a past tense verb in the lexical part and can also be explicitly allowed as an auxiliary verb in the grammar rules in S1, in a rule like

Vbar -> ‘was’ VBpast

In the lexical, we are already giving you the complete set of terminal symbols embedded in rules of the form “Tag -> word”. But most of the Tags are given as the tag “Misc” and one of your jobs is to replace the Misc. tags with a more meaningful tag whenever you need to parse the word.

In the structure part of the file, there is a simple grammar to start with. To extend this grammar, you can add new rewrite rules, you can add new phrase type symbols with their own rules. All of these rules can use the new tag symbols that you write in the lexical rules.

!Important: Since we are using a recursive descent parser, you must not include any left recursion in the rules. If there is, the parser will run for several minutes and then give an error that it has run out of space. If this happens, you must find a way to restructure the grammar so that there is no left recursion.

Running the parser

When you want to see what your grammar does, you can parse sentences with it. You can choose to run the small development program on the command line. Or you can copy/paste the small program into jupyter notebook.

Recommended development procedure

1. Run the parser and observe from the output which sentences have parse trees. Pick a sentence that does not have a parse.
2. If there are any words labeled Misc, try to label them with something else that is either a POS tag or similar. For example, we may use “Adj” for words that are being used as

adjectives. Go to the lexical rules and change that tag for that word, and any other words in the same class.
3. Add one or more rewrite rules to the structure rules. Create new non-terminal symbols as needed.
4. Reload the parser and check the parse trees!
5. Write down the sentence and the new rules that you created for your report.
6. Repeat as needed.

If you need ideas for grammar rules, put your sentence into the Stanford demo parser and figure out what kind of rules could generate that tree. Your rules cannot and should not try to get exactly the parses from the Stanford parser (see the Notes below), but do try to make sensible phrases, e.g. verb phrases should start with a verb. In general, you should give good phrases for noun phrases, verb phrase, prepositional phrases, and clauses (if any), but within the phrases your structure may vary.

Part 1: Writing grammar rules
You should write grammar rules that parse all of the sentences in sentences.txt. In addition, you should choose two additional sentences from the challenge sentences file.

Notes:
In this grammar, the prepositional attachments may be wrong. That's o.k. You can't really fix these without statistical parsing.

Restrictions:
Do not treat all the verb modifiers as if they were just verbs.
Prepositional phrases should start with a preposition, both noun and verb phrases should not start with a preposition.
Noun phrases should also not start with most types of verbs (the one exception is a gerund, or what our lexical rules call present participles, like "riding".
Verb phrases almost always start with a verb.

Part 2: Exemplar Sentences
Given the grammar that you wrote for part 1,

- make up a sentence that uses some of the same words in the sentences that you already parsed and is an actual English sentence, but cannot be parsed by those rules. The point of this sentence is to demonstrate some type of phrase structure that this grammar does not parse. You can use challenge sentences to get ideas of what can't be parsed yet by your grammar, but you should generate a sentence which is not an actual challenge sentence.
- make up a string of words that should not be an actual English sentence, but your grammar will parse it. [If the grammar parses obvious non-English sentences, it is "over-generalization", i.e. it is not restrictive enough in the forms that are allowed. Note that we wouldn't want a grammar that overgeneralized too much (hence the restrictions above), but for the sake of simplicity in writing the grammar rules, we do allow some. Note that the Stanford parser also overgeneralizes.]

What to submit for Homework:

Part 1:
Each student should submit a report that includes the final grammar that you got with the original sentences in sentences.txt and your two challenge sentences. Then for each sentence,

- write down the sentence,
- show the parse tree and
- write down the rules that you added to Vocab.gr and S1.gr.
- Add a sentence or two that describes how the rules are applied to parse the sentence, including the significant rules from the grammar; one way is to discuss the parse and describe which part(s) are new for that sentence. You must also explain in your own words **how those rules are generating the subphrases**.

If you had any difficulties, you can describe them.

Part 2:
Give your two exemplar "sentences". For the first sentence, explain what part of the phrase structure is not represented by the grammar in part 1 and show the parser output with no parse
For the second "sentence", explain how overgeneralization allows this so-called sentence to be parsed and give the parse tree output from the parser.