**Natural Language Processing**

*Spring 2020*

Individual Programming Project #2

For this project, you will implement a parser. Your program will prompt the user to enter the name of a text file specifying a context free grammar (CFG) in Chomsky normal form (CNF). Various assumptions about the format of the CFG below will be specified below. The CFG will represent a phrase structure grammar, a.k.a. constituency grammar, for English. After reading the grammar, your program will allow the user to enter sentences (one at a time). For each sentence, your program will parse the sentence using the Cocke-Kasami-Younger (CKY) algorithm. If there are no valid parses, your program should output "NO VALID PARSES". If there are valid parses, all valid parses should be indicated using a format that will be specified below.

Your program may assume that the CFG specifying the grammar is already in CNF. Each line of the file will contain a single valid rule in one of two forms: A --> B C, where A, B, and C are non-terminals, or A --> w, where A is a non-terminal and w is a terminal (i.e., a word). All non-terminals will start with a capital letter or an underscore ('\_'). All terminals will start with a lowercase letter or digit. There will be no OR symbols. Lexical rules will be part of the grammar (each on its own line). The start symbol will always be S. Every line in the file will be valid.

Of course, when using your program, it would be nice if you could start with a CFG that is not in CNF. I am providing you with a program I wrote that converts a general CFG to CNF. I wrote the program using Python 3; the program does not rely on any special libraries. The program assumes that all non-terminals start with a capital letter (not an underscore) and all terminals start with a lowercase letter or digit. (The dummy non-terminals that it creates will start with underscores, guaranteeing that there are no conflicting names.) My program allows OR symbols to be used in the original CFG, but the CFG in CNF that it creates will not use them. Optional constituents (surrounded by parentheses) are not allowed in either grammar. My program ignores blank lines (containing only whitespace) and lines starting with '#' (which can be used to indicate comments). Note that *your program does NOT have to indicate parses according to the original CFG*, only according to the CFG after conversion to CNF (this is what it will read and process).

On the class website, I will post my program to convert a general CFG to CNF, a sample grammar (a general CFG), and the grammar after conversion to CNF. To create this sample grammar, I started with the L1 grammar from the textbook, and then I added a few additional grammar rules and some extra words to the lexicon. I also added some invalid rules to test the syntax error checking of my conversion program, but I do not guarantee that it will detect all possible types of errors. If my program detects any sort of syntax error in a line from the original CFG, it displays an error message and ignores the entire line; the program will then continue to process the other lines of the CFG. (I also tested my program on a more complex grammar, but complex in terms of containing various funky things that would never likely occur in a real natural language grammar.) When I test your programs, I'll use a much larger grammar. I would appreciate if students test out my program on other grammars that you find or create. *Please let me know if you discover any bugs!* Feel free to share grammars that you create, and I'd also appreciate if you send me grammars that you think would help me evaluate programs.

Again, your program will only be tested using a CFG in CNF. After reading the grammar in CNF, your program should allow users to enter sentences, one at a time. You may assume that the user will only enter sentences with words complying to our programs' assumptions (that is, all words will start with a lowercase letter or a digit). For example, they might type "i book the flight from houston". If you want to allow the user to type sentences such as "I book the flight from Houston.", your program may include a pre-processing component that strips punctuation and converts all letters to lower case, but this is not required. (Note that we are assuming that grammars will NOT handle punctuation. Since we are assuming that all terminal symbols start with lowercase letters or digits, something like a period would not be a valid terminal symbol.) After processing each sentence, the program should either output "NO VALID PARSES" (only if appropriate, of course), or it should display all valid parses using a format that will soon be described. If the user types "quit", the program should end.

For each sentence entered by the user, the program should use the textual format known as bracketed notation. The textbook shows an example in Equation 12.1, on page 207 of the current, on-line draft of the textbook, as of the time I am typing this. (In class, we briefly looked at similar notations, demonstrated in Figure 12.7 and used by the Penn Treebank. These are not the formats I am asking you to use for the project, but the format shown on the left is very similar to bracketed notation. The format shown on the right would be more common if a POS tagger was applied before the parser, but we are assuming that lexical rules are treated the same as other rules.) Using bracketed notation, each constituent (i.e., type of phrase or POS) is enclosed in brackets, starting with the type of constituent followed by the sequence of sub-constituents. For example, for the sample sentence mentioned above, using the provided sample grammar (in the original CFG format, not in CNF), there would be (at least) three valid parses (I created these by hand, please let me know if you catch any mistakes):

* Parse 1: [S [NP [Pronoun i]] [VP [Verb book] [NP [Det the] [Nominal [Nominal [Noun flight]] [PP [Preposition from] [NP [Nominal [Noun houston]]]]]]]]
* Parse 2: [S [NP [Pronoun i]] [VP [Verb book] [NP [det the] [Nominal [Noun flight]]] [PP [Preposition from] [NP [Nominal [Noun houston]]]]]]
* Parse 3: [S [NP [Pronoun i]] [VP [VP [Verb book] [NP [det the] [Nominal [Noun flight]]]] [PP [Preposition from] [NP [Nominal [Noun houston]]]]]]

This is a good place to mention that I have not programmed a CKY parser myself yet, but I will, along with the rest of the class! I'll be able to compare my output against yours, and it will help me to answer questions that come up if there are any unforeseen difficulties. I will strive to finish my parser at least one week before the due date. Remember that the parsers will be tested only using grammars in CNF; the examples above are just provided to explain bracketed notation.

Finally, I am allowing one optional simplification: For a maximum grade of 80 (i.e., a loss of 20 points), you may implement a recognizer instead of a full parser. If you choose this option, your program should always output either "NO VALID PARSES" or "VALID SENTENCE".

E-mail your project to CarlSable.Cooper@gmail.com (I'm moving back to my Cooper Gmail account to avoid problems receiving Python files). The program is due the night of Tuesday, April 21, before midnight.