Natural Language Processing - Project 3
CSE 398/498-013
Prof. Sihong Xie
Due Date: 11:59pm, Nov 23 , 2017

1 Problem Statement

Given a PCFG in CNF, generate and parse a random sentence.

A PCFG Grammar The grammar is given in the file *data/grammar.gr* and you can use any text editor to view its contents. In the file, there are empty lines, comments (starting with #) and productions. Lines 20 - 23 with one comment line, one empty line and two productions look like

```
# Rules for creating full sentences.

1 ROOT S .
1 ROOT S !
```

Each production (e.g., *1 ROOT S .*) has 3 fields: a number indicating its frequency ("1" in this example. It is NOT the production probability), a left-hand-side (LHS, "*ROOT*" in this case), a **list** of symbols on the right-hand-side (RHS, "*S .*" in this case). There are three sets of symbols: 22 terminals (all in lower cases, including the period and exclamation), 5 pre-terminals (*Det, Adj, Verb, Prep, Noun*, only the first characters are in Upper case) and 5 terminals (*PP, NP, S, ROOT, VP*, all characters are in Upper case).

Generation Starting from the symbol ROOT in a list, expand any non-terminals and pre-terminals in the list, using one of the proper productions (picked proportionally to its probability), until there is no more non-terminals and pre-terminals remained in the list. This can be done using Depth-First Search (DFS).

Parsing The CYK algorithm can parse the sentence you generated, using the same PCFG. The algorithm is quite simple and the textbook shall give you enough ideas. Pay attention instead to highly efficient and clear data structures.

2 Exercises

- How can you modify the production frequencies so that longer sentences can be generated? Explain how and why.
- Is every sentence generated by the grammar can be parsed by the CYK algorithm using the same grammar? Give your intuition.

3 Experiment

Download and unzip the zip file (project3.zip) from Piazza. Implement the empty functions in Generator.java and Parser.java under *src*. Data structures and various utility functions for

handling grammars is implemented (in Grammar.java and RHS.java) so you can focus on the above two tasks, for which you need to design algorithm and data structures. The input to the generator is just the grammar file. The output is a parse tree used to generate a sentence, such as

```
(ROOT (S (NP (Det the) (Noun president)) (VP (Verb understood) (NP (Det every) (Noun pickle)))) !)
```

This output is redirected to the file *results/sentence.txt* (so you will not see this output). The file is then read and piped into a perl program (*src/prettyprint.pl*) to print the parse tree. Next the file is piped into your parser to produce a tree, which is redirected to the file *results/parse.txt* Note that you may (or may not) get more than one tree. One and only tree needs to be output.

Two scripts *build_java.sh* and *run_java.sh* will be used to compile and run your programs. Make sure you can run them on a Sunlab machine.

4 Deliverables

Please describe, clearly and succinctly, your algorithms and data structures for your generator and parser in the file *report_p3.pdf*. Within the same file, show the two parse trees printed on your command line. The file needs to be put in the root dir of the folder named <your Lehigh id>_p3, which is zipped into <your Lehigh id>_p2.zip and submitted to coursesite. Don't submit the project from your IDE. Don't alter the folder names and organizations in the unzipped folder.

5 Grading

The following aspects of your submission will be graded:

- Functionality: make sure I can compile your codes and produce desirable results using the two bash scripts.
- Readibility: how heavy your codes are commented and how clear you document your algorithms and data structures.
- Speed: there is no big data, but your implementations shall reflect your effort to achieve the best time and space complexity.