Elie Weintraub NLP Project #2 Write-up 5/9/14

**I. Overview**

For the second NLP project I implemented a syntactic parser in Python. The project consists of 2 .py files (CKY.py and CKY\_driver.py) as well as several text files (.cfg and .txt). The first .py file implements the CKY module. It defines the various classes necessary to perform the syntactic parsing. The second file implements a driver program. It uses the defined classes to parse a list of one or more sentences. The driver program takes a context-free grammar and a sentence file (i.e., a file of sentences to parse). It then converts the grammar to a form usable by the CKY algorithm, and finally applies the CKY algorithm to each sentence in the sentence file. The output of the program includes the converted form of the grammar as well as an output file containing a text representation of the sentence parses. Additionally, NLTK is optionally used to display a user-friendly version of the parse trees.

**II. Contents**

* CKY.py – The CKY module
* CKY\_driver.py – The driver program
* grammars\ - A folder containing sample CFG files
* input\ - A folder containing sample input files for each of the sample CFG files
* output\ - A folder containing sample output files for each of the sample input files
* reference\_results\ – A folder containing a file with results from the NLTK

**III. CKY.py**

As mentioned above, this file defines the various classes necessary to perform the syntactic parsing. Specifically the following classes are defined: a Tree Class, a Production Class, a Grammar Class, and a Parser Class. The Parser Class defines both a CKY Recognizer as well as a CKY Parser. The CKY Recognizer doesn’t include back pointers and so simply outputs whether or not the grammar accepts the sentence. The CKY Parser includes back pointers and so outputs all valid parses. Both the CKY Recognizer and the CKY Parser use an augmented form of the CKY algorithm that allows for the CFG to include unit productions. Thus the CFG does not need to be in strict Chomsky-Normal Form (in which productions are only of the form A→B C or A→ a) in order to run the CKY Recognizer or CKY Parser, but can instead be in flexible Chomsky-Normal Form (in which productions are only of the form A→B C, A→B, or A→ a). Seeing as the CKY algorithm needs the CFG to be in this flexible CNF, the Grammar Class includes a method to convert an arbitrary CFG to flexible Chomsky-Normal form. Also, to enable output to be in the form of a parse tree corresponding to the original grammar, the Tree Class includes a method to convert a parse tree using symbols from a CFG converted to flexible CNF to a parse tree using the symbols from the original CFG.

**IV. CKY\_driver.py**

As mentioned above, this file consists of a driver program that uses the defined classes to parse a list of one or more sentences. The program prompts the user to specify several parameters. These include:

* the file containing the sentence or sentences to parse
* the file containing the representation of the CFG
* whether or not the CFG is in (flexible) Chomsky-Normal Form already
* the name for the output file that will contain the grammar in (flexible) CNF (if conversion was necessary)
* the name for the output file containing string representations of the parse trees
* whether or not to display a graphical representation of the parse trees

Once all parameters have been specified the program uses the CKY Parser to parse all sentences in the input file. If the user chooses to have a graphical representation of the parse trees displayed and more than one valid parse exists for a given sentence, then after displaying each tree the user will be further prompted as to whether s/he wants to see another parse for the given sentence, or proceed with the next sentence (if it exists). Either way, all parses will be stored in text-form in the output file specified by the user.

**V. Grammar Files**

The grammar files are text file representations of Context-Free Grammars. They specify all the productions of the grammar. Additionally, they can optionally explicitly specify the start symbol of the grammar. If not explicitly specified, the start symbol is taken to be the left-hand side of the first production specified. These files should follow the following formatting rules:

* Lines beginning with a # are treated as comments and ignored by the program
* To define the start symbol as START\_SYMBOL include a line of the form ‘%start START\_SYMBOL’
* To define a production, separate the elements of the production with white space characters and separate the LHS and RHS with an arrow (‘-> ’) itself surrounded by whitespace. Thus an example production could be of the form

‘LHS -> RHS\_1 RHS\_2 RHS\_3’

* Additionally the ‘|’ shorthand can be used to define multiple productions simultaneously. For example ‘LHS -> RHS\_1 RHS\_2 | RHS\_3’ defines the 2 productions ‘LHS -> RHS\_1 RHS\_2‘ and ‘LHS -> RHS\_3’
* Finally, terminal symbols should be surrounded in double-quotes.

The ‘grammars/’ folder included in the project includes some sample CFGs. atis.cfg and toy.cfg were obtained from the NLTK while I created L1.cfg based on the L1 grammar detailed in *Speech and Language Processing* (Jurafsky and Martin).

**VI. Sample Input Files**

The folder ‘input/’ included in the project contains sample input files for each of the sample CFGs included in ‘grammars/’. Input files in general should consist of one or more sentences to parse, each separated by a newline.

**VII. Sample Output Files**

The folder ‘output/’ included in the project contains sample output files produced by the CKY\_driver.py program.

**VIII. Reference Results**

The folder ‘reference\_results/’ included in the project, contains a file containing reference results provided by the NLTK for the sentences in atis\_sentences.txt. Each line of this file includes the number of parses produced by the NLTK followed by a colon followed by the sentence being parsed. This file was used to evaluate my project by comparing the results my project produced to those stated in this file.

**IX. Results and Evaluation**

The results of the project were evaluated using three grammars. Two toy grammars were used (toy.cfg and L1.cfg) as well as one large grammar (atis.cfg). For each of the toy grammars results were evaluated by manually looking at the output and confirming that the parses were correct. For the large grammar correctness of results was additionally evaluated by comparing the number of parses my project produced to those stated in the reference results file. Finally, it was manually confirmed that the conversion to and from flexible CNF was done correctly.

**X. Requirements and Use**

The program requires Python 2.7. Additionally, the Natural Language Toolkit (NLTK) package should be installed. If the requirements are met the program can be run in IPython using the command "run CKY\_driver.py" or from the Linux terminal (or Cygwin) using the command "python CKY\_driver.py”.

Once run, the program will prompt the user with various prompts and the user need simply follow the prompts.