Classes and methods

CKY.py

A 'CKY' class is implemented:

- **CKY:**
  __init__(self,grammar_cnf): a CKY object is instantiated with a cnf grammar.
  A dictionary *_rhs_lhs* is created from the grammar, to access the left-hand sides of the rules from the right-hand sides.

  __parse(self,sentence): given a sentence, creates and fills in the CKY chart.

  recognize(self,sentence): returns *True* if the sentence is grammatical, *False* otherwise.

  number_of_trees(self,sentence,i,i_plus_b,A): returns the number of trees for a given entry of the CKY chart of a sentence (A(i,i+b)), if there are any.

  __count_trees(self,i,i_plus_b,A): recursively counts the number of trees underlying a given grammatical entry (A(i,i+b)): for a given A, the number of underlying trees is the sum (over all the productions A → B C) of the number of trees underlying B times the number of trees underlying C. If the rule is preterminal, there is only one underlying tree.
  Returns the number of trees underlying A(i,i+b).

  trees(self,sentence): returns a set of *ImmutableTrees* for a given sentence, if it's grammatical.

  __write_trees(self,i,i_plus_b,A): recursively enumerates and writes down the trees strings *for a given grammatical entry (A(i,i+b))*.
  Returns a set of the trees strings.

#NOTE: The methods that start with '__' are private methods.
##NOTE: We can count the number of trees with len(trees). I implemented the *'number_of_trees'* method to compute the number of parse trees for an entry A from the chart with backpointers, without having to explicitly compute the parse trees. The code runs in approximately 1/4 of the time needed when using len(trees). Given the same number of recursions, each call to *__count_trees* runs in linear time, each call to *__write_trees* in cubic time.

myGrammar.py

def convert_to_cnf(grammar): given a non-lexical grammar, returns and equivalent grammar in chomsky normal form, if the start symbol never appears in the right-

hand side of the rules.

def eliminate_more_than_2_nonterm(grammar): given a non-lexical grammar, returns an equivalent grammar with no more than two non-terminals on the right-hand side of each rule, if the start symbol never appears in the right-hand side.

def eliminate_unary_rules(grammar): given a non-lexical grammar with no more than two non-terminals on the right-hand side of each rule , returns and equivalent grammar in chomsky normal form.

How to run the program

You can run the program following these steps:

1) Open CKY.py in python3.

2) A list of ATIS test sentences with tab-separated numbers of parse trees is written to the ATIS_output.txt file. In case the sentence is not recognized from the grammar, 'There are no grammatical trees for this entry.' will be printed instead of the number of trees.

3) The parse trees for first ATIS test sentence with 4 parse trees are drawn.

Execution

1) The ATIS corpus is downloaded from nltk.data, and the sentences are extracted.

2) The file "atis-grammar-cnf.cfg" should be in your current directory. If not, it will be automatically downloaded.

3) The ATIS cnf grammar is read from the "atis-grammar-cnf.cfg" file as an nltk.grammar.CFG object.

4) The original grammar is nonlexical, but not in cnf. This means that there are no empty rules, and all pre-terminal rules are unary. Moreover, the start symbol never appears in any right-hand side of the rules. To convert it to a cnf grammar, we only need to 1) get rid of non pre-terminal rules with more than two non-terminal symbols on the right-hand side, and 2) get rid of unary rules.

5) We call the *convert_to_cnf* function from the myGrammar module on the original grammar. First, the function *eliminate_more_than_2_nonterm* is called on the original grammar, then the function *eliminate_unary_rules* is called on the

output of *eliminate_more_than_2_nonterm*. The cnf grammar is returned.

6) An ATIS_output function is defined:
- For the given cnf grammar and a filename, a CKY object is instantiated with the grammar: a dictionary *rhs_lhs* is created, to access the left-hand side from the right-hand side symbols.
- The file is opened for writing with the given filename.
- For each sentence of the ATIS corpus, the sentence is written as a single string, such that there are no spaces before punctuation symbols or at the beginning of the sentence.
- The function *number_of_trees* is called on each sentence for the entry START_SYMBOL(0,lenght of sentence) of its CKY chart. The chart is filled in with the CKY algorithm by the *__parse* function. In each cell (i,i+b), the non-terminal symbols (A) are stored, together with the corresponding terminal symbol for preterminal rules, the B and C non-terminal symbols and the index (i+k) for the non-preterminal rules (where B and C are the right-hand sides of each A → B C, and (i+k) serves as backpointer to the cells from which B and C are originated). If the cell (0,lenght of sentence) is empty or doesn't contain the start symbol, 'There are no grammatical trees for this entry.' is returned. Otherwise, the function *__count_trees* recursively counts the trees underlying the entry START_SYMBOL(0,lenght of sentence). In this case, the number of trees is returned.
#NOTE: During the iteration, we assign the first sentence with 4 parse trees to the variable sentence_of_choice.
- A list of ATIS test sentences with tab-separated numbers of parse trees is written to the output file. In case the sentence is not recognized from the grammar, 'There are no grammatical trees for this entry.' will be written instead of the number of trees.
- The parse trees for first ATIS test sentence with 4 parse trees are drawn.

7) The ATIS_output function is called both on the cnf ATIS grammar (as read from the atis-grammar-cnf.cfg file) to write the output file ATIS_output1.txt, and on the cnf ATIS grammar (as converted from the original grammar using the myGrammar module) to write the output file ATIS_output2.txt. We obtain the same results (compare the output files and the trees).