ASSIGNMENT 2

COMP 599, Fall 2015

Due: October 20^{th}, 2015 in class. No late assignments accepted.

You must do this assignment individually. You may consult with other students orally, but may not take notes or share code, and you must complete the final submission on your own.

Question 1:	20 points
Question 2:	40 points
Question 3:	40 points
	100 points total

Assignment

Question 1: POS Tagging (20 points)

Train a first-order (i.e., the probability of a tag depends only on the previous tag) HMM part-of-speech tagger by hand on the following training corpus. Find the MAP estimate of the parameters of the model using add-1 smoothing.

```
That/C that/N is/V , is/V .
That/C that/N is/V not/N , is/V not/N .
Is/V that/N it/N ?
It/N is/V .
```

Ignore capitalization differences and punctuation. They are only there for readability purposes. There should thus be a tagset of size 3, and a lexicon of size 4. Your model should contain the following parameters: Π (initial state probabilities), A (state transition probabilities), and B (emission probabilities).

Next, run the Viterbi algorithm (by hand) to obtain a POS tagging for the following test sentence. Show the trellis cell values and your calculations.

```
Not is not that is .
```

Question 2: Grammar for Travel Planner (40 points)

In this question, you will develop a context-free grammar for a travel planner. The intended use case of the CFG is a smartphone app which takes in multiple declarative sentences and extracts some semantic information from them in order to create an itinerary of a trip.

For example, given the input sentence

```
Jackie is flying to Portugal tomorrow .
```

it should be possible to derive the following information from the correct parse of the sentence[1]

```
(TRAVELLER Jackie) (MODE is flying) (DEST to Portugal) (DEPTIME tomorrow) .
```

[1] This is not a full, complete parse, just a partial parse to illustrate the semantic categories that can be read off from the parse tree.

In other words, the parse tree should contain constituents that are labelled by their semantic category, rather than by their syntactic category as in a standard syntactic parse.

Use the following list of semantic categories in your grammar:

```
TRAVELLER    The traveller(s) taking the trip
MODE         The mode of transportation (on foot, by bus, by plane, etc.)
DEP          The point of departure
DEST         The destination
DEPTIME      The time of departure
ARRTIME      The arrival time
```

Here are some sentences that your grammar should recognize:

```
Jackie is flying to Portugal tomorrow .
We are going to the park at 12pm .
I am travelling by bus to the zoo from my apartment .
On Monday, they will walk up Mount Royal .
```

The grammar should also encode some syntactic knowledge. For instance it should know what kinds of phrases tend to be expressed by which syntactic categories. A TRAVELLER will be expressed by a noun phrase, for example, and not a verb phrase. On the other hand a MODE of transportation will be expressed either by a verb phrase (e.g., *is flying*) or by a prepositional phrase (e.g., *by bus*). Your grammar should be aware of the basic sentence order of English declarative sentences (i.e., subject-verb-object, with other arguments following), and the fact that arguments dealing with time, manner, and place in English may have multiple orderings.

Use the following nonterminals to indicate grammatical categories:

```
S           sentence/clause
NP          noun phrase
VP          verb phrase
PP          prepositional phrase
N           noun (common or proper)
PR          pronoun
AUX         auxiliary or modal verb
V           verb
P           preposition
DT          determiner
ADV         adverb
```

You may add further non-terminal categories or subdivide them (e.g., N-transportation) as needed. Don't forget the lexical rules! Include enough lexical items such that each of the semantic categories can be expressed in at least three different ways (e.g., MODE might be expressed by *by bus*, *by car*, and *flying*).

Figure 1 shows an example of the full tree of the above sentence including both syntactic and semantic categories.

Write your grammar in a text editor using a predictable, computer-readable format, because you will need this file for the next question. For instance, here is one possible rule:

```
S -> NP VP
```

Here is another example of a set of four rules (here, they are lexical rules):

```
V-mode -> fly | flying | walk | walking
```

These are just examples, and are not necessarily the rules you want in your grammar! Ignore punctuation and capitalization in your grammar (just use all lower-case, except capitalize I and proper names).

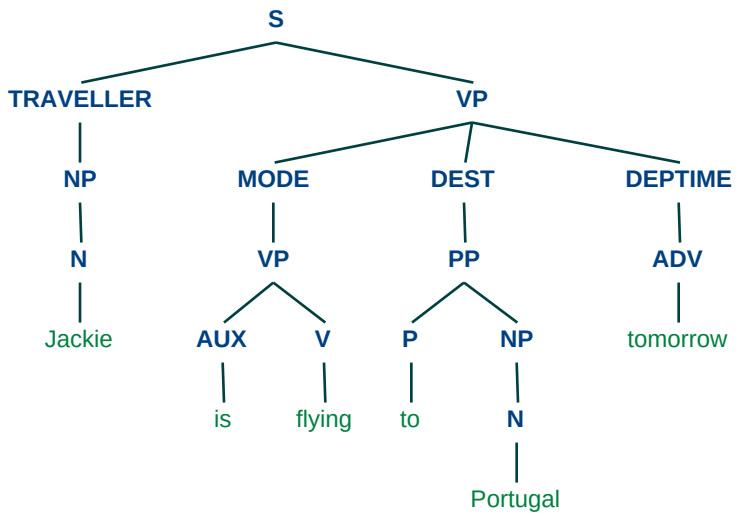

Figure 1: Example of a parse tree including syntactic and semantic annotations.

Submit your grammar on paper and as a text file. Show instances where your grammar succeeds, and other instances where your grammar fails by overgeneration and by undergeneration. In addition, answer the following questions in your response to the question:

1. What aspects of the English grammar are captured by your CFG?
2. What aspects of the English grammar are NOT captured by your CFG? How could you change your grammar to correctly capture it? Would this come at any cost to the extraction of semantic information from the parse tree?

This question is rather open-ended; your grammar will be judged on the following points:

- Whether you followed the specifications above (e.g. names of non-terminals, minimum number of lexical entries)
- Coverage in terms of different grammatical constructions
- Clarity of the grammar
- Whether you describe the overall flaws of the grammar clearly and succinctly (We expect that there will be many aspects of English grammar and the domain of travel planning that your CFG doesn't cover!)

You won't get extra points for having many additional lexical items that exhibit the same type of behaviour!

Question 3: Implement an Earley parser (40 points)

Read (J&M 2nd ed: 13.4.2; J&M 1st ed: 10.4) and implement an Earley parser for your grammar above. The Earley parsing algorithm is a dynamic programming algorithm, like the CYK parser, but it works top-down instead of bottom-up. That is, it begins from a starting non-terminal symbol, and expands it according to rules that would generate the words in the leaf nodes. The only existing code that you may re-use is the Tree data structure from NLTK, to help you with representing and displaying your trees. You must write your own parser from scratch otherwise, and may not consult pre-existing implementations of Earley parsers.

Apply your parser to your test cases above, and show several sample parses in a report on your parser. Verify that the parser under- and overgenerates on the cases that you came up with for the previous question.

We will run your code to verify your grammar and parser. Put your code in a file called *earleyparser.py*. In particular, running the following on the command line:

`python earleyparser.py <grammar_file>`

should read sentences in, one at a time, until it reaches the end of the file. The program should then print the parses to standard output using `pretty_print()`, with parses separated by an extra newline. For sentences that do not have a parse according to the grammar, it should print the sentence back out, unchanged.

Running the following command:

`python earleyparser.py draw <grammar_file>`

should instead read in one sentence (until a newline character), parse it, and then draw it to screen using `draw()`.

Hint: It might help to complete Question 3 before Question 2, using a simple dummy grammar to start with. Then, you can use your own parser to develop and debug your grammar.

What To Submit

On paper: Submit a hard copy of your solutions to Question 1 and 2 (including the grammar), as well as the report part of Question 3 in class.

For the programming part of Question 3, you should submit one zip file with the grammar file and your source code to MyCourses under Assignment 2.