Rule-Based Part of Speech Tagging
Due Oct. 29th, 2014

For this assignment you will get to explore rule-based systems and their application to part of speech tagging. Rule-based systems have seen much success applied to NLP, with most focus being on systems that understand the meaning of text. However, much of that is beyond the scope of this course, so we will narrow our focus to part of speech tagging. Early part of speech taggers were primarily rule-based, however nowadays they have lost favor with the computational linguistics community to stochastic methods that are much more quick and accurate. But, as Math-M 464 is not a prerequisite for this course, we'll be focusing on rule-based systems. Rule-based systems typically consist of two main components, the knowledge base and the inference engine. You will be implementing the knowledge base and integrating it with an inference engine we have provided for you.

The Knowledge Base

The knowledge base of your rule-based system will contain all of the factual information needed to understand and tag sentences given to your system. To illustrate this we will consider the simple sentence *"Time flies like an arrow"*. We want two separate sets of rules when organizing knowledge of this domain:

- *Tags associated with words:* Certain words are likely to correspond to certain tags. Consider the word *time*, in the example sentence it is a noun; however in the sentence *"Will SOMEONE PLEASE time how long it takes me to do 37 pushups"* time is a verb.
- *Tags associated with tags:* Part of speech tags for a particular word are largely determined by the part of speech tag of the immediately preceding word. Verbs are often preceded by nouns, nouns often preceded by adjectives, etc.

As Ling-L 545 is not a prerequisite for this course, we will provide to you all of the tagging relationships you need in a text file.

Rules and Facts

Rules in your knowledge base will consist of three parts.

- *Name:* Name of your rule.
- *Antecedents:* IF conditions of your rule.
- *Consequents:* THEN conditions of your rule.

Facts will pertain to the sentence given to your system. As an example, consider the sentence *"Mike is running."*, the facts we could produce from this sentence would be:

- *"Mike"*
- *"is"*
- *"running"*
- *"BEGIN_SENTENCE precedes Mike"*
- *"Mike precedes is"*
- *"is precedes running"*
- *"running precedes END_SENTENCE"*

Implementation and Integration

In order to ensure the inference engine works with your rules and facts, the list of antecedents and consequents in your rules as well as your facts MUST be in the following format. An individual antecedent, consequent, and fact must be a string. The list of antecedents, consequents, and facts are lists of strings. Your rules may contain variables that allow less restrictive matching. In the below example you can see that *?word* has been specially marked with a *?*. That symbol indicates to the inference engine that it can be matched with any single word. An example of this is outlined below.

```
name = "Mike-Tag"
antecedents = ["Mike"]
consequents = ["Mike = Proper Noun"]
rule1 = Rule(name, antecedents, consequents)

name = "BeginSentence-Tag"
antecedents = ["BEGIN_SENTENCE precedes ?word"]
consequents = ["?word = Noun", "?word = Preposition"]
rule2 = Rule(name, antecedents, consequents)

WorkingMemory = ["Mike", "is", "running", "BEGIN_SENTENCE precedes Mike", \
                              "Mike precedes is", "is precedes running", "running precedes END_SENTENCE"]

Rules = [rule1, rule2]

system = RuleBasedSystem(Rules, WorkingMemory)
system.generateInferences()

print system.workingMemory

>>>["Mike", "is", "running", "BEGIN_SENTENCE precedes Mike", \
               "Mike precedes is", "is precedes running", "running precedes END_SENTENCE", \
               "Mike = Proper Noun", "Mike = Noun", "Mike = Preposition"]
```

The keen observer may notice that in our working memory of facts known to the system, *"Mike"* has been assigned three different parts of speech. While this is fine to have in working memory, when we output the words with their associated parts of speech we won't want multiple parts of speech to be assigned to a particular word. For now, we will use a conflict resolution strategy to clarify this ambiguity, but in the next assignment we will do something more complicated.

In order for your program to work, you will need to make use of the RuleBasedSystem class. It has the ability to call on the inference engine to generate new facts that follow from your rules and working memory. A description of the class and how to use it is below.

- *Initializing RuleBasedSystem:* To begin you will want to create both a list of rules and a list of current facts in the format described above and pass those into the class. Observe the example below.

  ```
  RuleList = []

  rule1 = Rule("first-rule", ["?word1 precedes ?word2", "?word2 = Verb"], ["?word1 = Noun"])

  RuleList.append(rule1)

  WorkingMemory = []

  fact1 = "Mike precedes went"
  fact2 = "went = Verb"

  WorkingMemory.append(fact1)
  WorkingMemory.append(fact2)

  yourSystem = RuleBasedSystem(RuleList, WorkingMemory)
  ```

- *Using the inference engine:* Calling the inference engine is then quite simple. Once called, it will append to your working memory all of the new facts the inference engine has generated. We continue from the previous example.

```
yourSystem.generateInferences()

print yourSystem.workingMemory

>>>["Mike precedes went", "went = Verb", "Mike = Noun"]
```

Your Task

Your system will have a function named *tagSentence* that takes in a sentence without any punctuation or capitalizations (the sentence will only consist of lower-case words) and return all of the words with their associated tags. An example of how your program will be expected to run is shown below (assumes all rules have been defined earlier in the program).

```
sentence = "mike is running"

taggedSentence = tagSentence(sentence)

print taggedSentence

>>>["mike = Proper Noun", "is = Verb", "running = Verb"]
```

As Ling-L 615 is not a prerequisite for this course, your system DOES NOT have to handle every possible combination of words in existence, however there will be test cases where words will not be known to your system. We are providing you with a test file that you may use to check your code. If your code passes this test file then your program will likely get a good grade (barring any cheating of course). The relationships we provide to you will dictate how words are assigned parts of speech. Also, the relationships may have your system assigning multiple parts of speech to a single word, which is not something a part of speech tagger should do. To resolve this issue, choose one or more conflict resolution strategies for rule execution. Your end output should assign ONE part of speech to each word in the sentence.

Provided Files

- *Tools.py:* Contains helper functions for converting a sentence to an array with all of the individual words of the sentence and for doing the reverse conversion of an array to a sentence.
- *InferenceEngine.py:* Contains the inference engine for the rule-based system. You don't really have to mess with this file. It took me a really long time to write though, so I'd hope you would at least glance at it. In fact, do look at this file. I spent a lot of time on it and you should appreciate my efforts.
- *KnowledgeBase.py:* Contains classes that you need to use to define your rules and your rule-based system. The classes are created so that they work with the inference engine, so make sure you use them.
- *Relationships.txt:* Contains all of the information that you will need to define your rules.

Grading

- ***50% of grade:*** *KnowledgeBase.py* extended to include all of the rules your system needs to assign part of speech tags to sentences.
- ***50% of grade:*** A function in *KnowledgeBase.py* named *tagSentence* that takes in a sentence and outputs all of the words with their associated tags.
- ***Bonus 10%:*** Modify your program so that when run it prompts the user for an arbitrary sentence (including capitalization and puncuation) and assigns parts of speech to the words of the sentence.

- ***Bonus 10%:*** Extend your rules to be able to handle conjunctions (i.e. and, but, or, etc.).

How to Submit

One file named *KnowledgeBase.py* containing your implementation of a rule-based part of speech tagger in a directory in your repository named "hw4".