Practical 6

Week 7

Artificial Intelligence Summer 2019

In this exercise sheet, we shall attempt to make use of the natural language processing (NLP) tookit within Python. The most popular library is NLTK (Natural Language Toolkit). But before the toolkit can be used, we shall do a few exercises that will help us understand how is natural language is represented within the Python programming environment.

Python uses the most intuitive way to represent a natural language sentence. Consider a sentence such as: "The child plays the fool". This sentence can be represented in the form of a graph consisting of nodes connected by labelled edges. Normally, the words are the nodes and the edges are the links between the words that provide the phonology associated with each node. The nodes can also be labelled. This is illustrated as an example below:

In the above diagram, the words are nodes listed as W1,……,W5 and the edges are links depicting the phonology such as Subject, Object and Determiner. Note also that it is important to distinguish between nodes and their labels. This enables us to differentiate between the two occurrences of "the" in the graph above, corresponding to the two nodes W1 and W4.

1. Creating a graph:
   A graph is represented using a dictionary. Let us start with a graph with no nodes or edges:

```
g = dict()
```

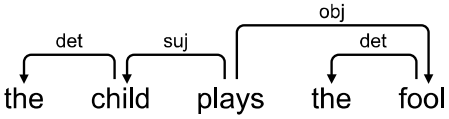
   The nodes are dictionary keys. The value corresponding to key v is a pair (a, sucs) made up of a label a and list of its successors sucs of labeled edges starting from v. Let us add a node 'W1' labeled *"the"* to g:

```
g['W1'] = ('the', [])
```

   Now, add a second and a third node, with the edges that connect them:

```
g['W2'] = ('child', [])
g['W3'] = ('plays', [])
g['W3'][1].append(('suj', 'W2'))
g['W2'][1].append(('det', 'W1'))
```

   Exercise 1: Implement the above code. Then add a code snippet to print the graph (not graphically).
   Exercise 2: Append a few more lines to the above code to completely represent the above sentence. Print out the final graph and confirm that it correctly reflects the structure of the natural language sentence as shown below:



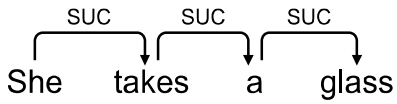
2. In practice, it is easier to use constructors as follows:

```
def add_node(g, u, a):
  #Add a node u labeled a in graph g
  g[u] = (a, [])

def add_edge(g, u, e, v):
  # Add an edge labeled e from u to v in graph g
  if (e, v) not in g[u][1]:
    g[u][1].append( (e, v) )
```

Exercise 3: Use the above constructs to recreate the graph for the above sentence

3. It is possible to segment a given sentence into words. This is an important step that is required for further logical reasoning and deduction in FOPL. The segmentation is represented as a "flat graph" connecting words in order as shown below.



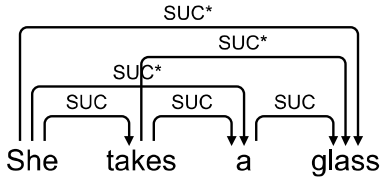
As can be seen, an edge 'SUC' has been added between each word and its successor. This is usually implemented with the help of the NLTK segmenter as follows:

```
import nltk
word_list = nltk.word_tokenize("She takes a glass")
word_graph = dict()
for i in range(len(word_list)):
    add_node(word_graph, 'W%s' %  i, word_list[i])
for i in range(len(word_list) - 1):
    add_edge(word_graph, 'W%s' % i, 'SUC', 'W%s' % (i + 1))
word_graph
```

And the output would be shown as:

```
{'W3': ('glass', []), 'W1': ('takes', [('SUC', 'W2')]),
'W2': ('a', [('SUC', 'W3')]), 'W0': ('She', [('SUC',
'W1')])}
```

Exercise 4: Finish constructing the following flat graph so that there is a 'SUC*' edge between each word and one of its distant successors. For example, the chain "She takes a glass" will be transformed as follows:



4. So far, node labels have been limited to their phonological form, i.e. a string of characters. Richer forms of structure, namely feature structures, may be required. Once again the same dictionary can be used :

```
fs_plays = {'phon' : 'plays', 'cat' : 'V'}
```

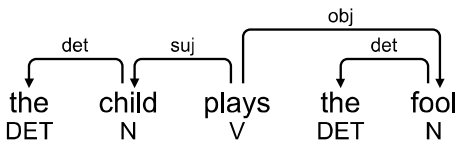
Here, the `fs_plays` dictionary designates a feature structure with two features, the phonology (denoted as 'phon') and its category 'cat'. In this particular case, the value of 'phon' and 'cat' are 'plays' and 'V' respectively. To find the category 'cat' of the feature structure `fs_plays`, we apply:

```
fs_plays['cat']
```

to get the output:

```
V
```

Exercise 5: Use the above example to create a graph for the sentence, "The child plays the fool". Start by creating nodes which will correspond to every word in the sentence. Use the `add_node` constructor shown on the top of Page 2. Name each word as W1, W2, W3..... and so on. Every node will have a label which would be the feature structure such as `{'phon' : 'plays' , 'cat' : 'V'}`. Possible categories within the feature structure are Determiner as 'Det', Verb as 'V' and Noun as 'N'. Next, add edges using the `add_edge` constructor shown on Page 2. Your final graph should represent the following:



Try to print out the graph.
P.S: Do not forget to initialise your graph as a dictionary.

5. The above can be easily implemented using the "Part Of Speech" (POS) labelling found in NLTK. The code might look like

```
import nltk
word_list = nltk.word_tokenize("She takes a glass")
tag_list = nltk.pos_tag(word_list)
feat_list = [{'phon':n[0], 'cat':n[1]} for n in tag_list]
t_graph = {'W%s' % i : (feat_list[i], [])
           for i in range(len(tag_list))}
for i in range(len(tag_list)-1):
    add_edge(t_graph, 'W%s' % i, 'SUC', 'W%s' % (i+1))
t_graph
```

Exercise 6: Try to understand the information contained in `word_list`, `tag_list` and `feat_list`. How does that correlate to the graph created in Exercise 4?

-END-