CS6770:Knowledge Representation & Reasoning

Assignment Report

Tapish Garg : MM17B034

Goal

Construct a FOL Representation of the corresponding natural language sentence using any Programming language.

Approach

The natural language sentences will have some rules and grammar that they follow. We have made a framework where the algorithm will remain same and we can more rules to our convenience.

So first we defined a **rules** which will have 4 components.
First component will the english sentence with the PoS tags which will act as pattern for sentence matching of input. We have used the functionalities of regex library which help to match the our input sentence with the rule. We have also implemented the optional property **"|"** of regex which gives us the ability to match multiple choices for a particular rule.
Example - \w+/**(NN|NNP|NNS)**\sis/VBZ\s**(a|the|an)**/DT\s\w+/**(NN|NNP|NNS)**

Second component will contain the location of predicates which are present for a particular rule.

Third component will contain the location of constants which are present for a particular rule. It may be the case that there are no constants for the rule.
Above first three components are defined in **rules.txt** file.

Fourth component will contain the logic in which the natural language sentence will be represented as FOL. This component will have placeholders which will be filled by predicates and constants. This component is defined in **rules.py** file.

One **Utilities.py** file is there which helps us in achieving small tasks. It contains following functions:
1) **getPosTag(s)** - This function receives string as an input and will generate the PoS tags for the string. Finally it will return a string with PoS tag after each word.

Input - Maths is a course
Output - Maths/NNP is/VBZ a/DT course/NN

2) **lemmatize(token)** - This function will help to lemmatize a word.
Input - courses
Output - course

3) **get_inputs(path)** - This function will take path of text file of input English sentences for which we need to construct FOL representation. The output of this will be an array of English input sentences as string.

4) **get_rules(path)** - This function will take path of rules text file. In rule text file each line corresponds to a rule. So this function will return the English, predicate and constant components of rule in an array format.
Input - \w+/(NN|NNP|NNS)\sis/VBZ\s(a|the|an)/DT\s\w+/(NN|NNP|NNS) 4 1
Output - {'rule': '\\w+/(NN|NNP|NNS)\\sis/VBZ\\s(a|the|an)/DT\\s\\w+/(NN|NNP|NNS)', 'predicates': [4], 'constants': [1]}

5) **find_predicate_and_constants(txt, rules, all_predicates)** - This function is the heart of the problem. It will take following the inputs:
First argument: input sentence that is generated after getPosTag function.
Second argument: array of rules that are defined by us.
Third argument: list of all predicates that we have put. This list of predicates can be extended to our benefit by manually identifying the predicates that are generally contained in our input sentences.
This function first matches the input text with the relevant rule and then give list of identified predicates, constants and rule number.

6) **get_all_predicate(path):** This function will make the list of all correct form of the predicates. It will take the path of predicates.txt file. This all predicates list helps to get the correct form of predicates.

Algorithm

First we find the out the PoS tags of the sentence and make into a format that will be used for matching the English component of rules that we have defined.
Input sentence - Maths is a course
Output sentence - Maths/NNP is/VBZ a/DT course/NN

After that we will use the matching function to find the out the rule that matches with the newly generated sentence in the above process. As the new sentence matches with the rule we will get the predicates, constants and rule number for the sentence.
Input English - Maths/NNP is/VBZ a/DT course/NN
Rule English - \w+/(NN|NNP|NNS)\sis/VBZ\s(a|the|an)/DT\s\w+/(NN|NNP|NNS)

Rule Predicates - course
Rule Constants - Maths
Rule Number - 1

As we will get the rule number we can get the rule logic function that we have defined. By putting predicates and constants at the appropriate placeholders of the rule function.
Rule Logic - simple(predicates[0], constants[0])
Final FOL - course(Maths)

Assumptions

We have made the following assumptions:

1) We have assumed that for a particular rule, the length of sentence that will match will be same. That is why we have specified the index of predicates and constants in a rule.
2) We have made a list of correct form of predicates that will be present in the input text file. This list can be expanded by adding more predicates.
3) We have added the limited variety of rules. There can be more rules which can be added to rules.txt and rules.py file.

References

https://www.cse.iitk.ac.in/users/karkare/MTP/2014-15/naman2015logica.pdf