INDIAN INSTITUTE OF TECHNOLOGY MADRAS

CS6770 - Knowledge Representation and Reasoning
Assignment Report – Conversion of English language sentences to FOL

Viswajith V – CS11B028
Sriram V – CS11B058

April 26, 2014

1 INTRODUCTION

1.1 PROBLEM SPECIFICATION

Our problem was loosely specified as: **Given a set of natural language sentences (like a word problem) construct a corresponding FOL representation.** The expressive power of natural language significantly dwarfs that of FOL, and there are several situations where ambiguities will be rife. We have, therefore, chosen to restrict our attention to sentences which have an obvious translation to FOL. All categories of sentences which FOL can express have been handled by our code; however, due to the difficulty of 100% accurate parsing, it cannot be guaranteed that our code will work for all sentences. We believe, though, that it should work for most common cases.

1.2 APPROACH TO SOLUTION

First-order logic requires us to identify predicates and quantifiers from the sentences, and figure out which other words in the sentences are arguments for the predicates. Therefore, it requires *context-sensitive* information. However, for efficiency of parsing, we decided to stick to a *mildly context-sensitive* approach, which allows for incorporation of context-related

information while still being light enough to not incur too much processing overhead. We have therefore parsed our sentences using the **C & C CCG** [1] parser, which works with a subset of grammar knows as CCG or Combinatory Categorial Grammar [2]. With the information gleaned by parsing it with this grammar, we can figure out which sort of FOL format our sentence falls into to figure out the structure of our final output, and then go over the exact words to fill in the blanks, so to speak, and produce the desired output.

2 Brief primer on CCG

Combinatory categorial grammar (CCG) is an efficiently parsable, yet linguistically expressive grammar formalism. It relies on combinatory logic, which has the same expressive power as lambda calculus [3], but builds its expressions differently.
It classifies words into the syntactic categories, which can be inductively built-up as:

(a) (Base case) N (noun), NP (noun phrase) and S (sentence) are categories.

(b) (Recursive clause) If A and B are categories, then so are (A/B) and (B\A)

(c) Nothing else is a category.

Now, for the semantics:

- A/B means that the word will give you an A, if it receives the B it expects on the right, ie, the concatenation of A/B and B is an A.
- A\B means that the word will give you an A, if it receives the B it expects on the left, ie, the concatenation of B and A\B is an A.

3 Organisation of Code

3.1 Handling unary predicates

Unary predicates are assumed to be specified by a form of the word 'be'. We take the object of the verb as our predicate, and the subject as our argument. Example: "John is a man" will become $Man(John)$ and "Humans are animals" will become $\forall(x)Human(x) \rightarrow Animal(x)$ in FOL.

3.2 Handling basic binary predicates

There are several forms that binary predicates can come in; we used a reference guide to CCG[4] which showed the different forms sentences can take with two-predicate verbs and adjectives, and incorporated them into our code. For example, conversion to lambda calculus

[1] http://svn.ask.it.usyd.edu.au/trac/candc
[2] http://en.wikipedia.org/wiki/Combinatory_categorial_grammar
[3] http://en.wikipedia.org/wiki/Lambda_calculus
[4] http://www.cs.columbia.edu/ mcollins/6864/slides/ccg.pdf

can be seen by $(S\backslash NP)/NP : \lambda x.\lambda y.p(x,y)$. Hence, Alice saw Bob will become $saw(Alice, Bob)$ after substituting for saw with the grammar rule and binding Alice and Bob to the λs for x and y.

3.3 Existential quantifiers

Using the same framework described above, we augment it by inspecting the parts of speech of various nouns; the tagging is one in accordance with the Stanford tagger[5]. All common nouns in singular are existentially quantified. For example: "All boys like a girl" will become $\exists x(girl(x) \wedge \forall y(boy(y) \rightarrow likes(y,x)))$

3.4 Subject-verb agreement and universal quantifiers

By inspecting the subject-verb agreement, and whether the noun is question is a common noun or proper noun, we can decide whether to universally quantify the variable. We do this irrespective of the presence of a guiding term like all, because the presence of all tends to lead to ambiguities in the grammar we use. An example of this is presented in the above subsection, where boy is universally quantified and girl is existentially quantified.

3.5 Conjunction and dysjunction

Our code is built to take in a set of natural language sentences. Conjunction and dysjunction are identified only by the keywords *and* and *or*; we then exploit their distributive properties to pass them as separate sentences to the rest of our code; thus, in essence, it is handled by a sort of wrapper. The sentences are then combined with the appropriate symbol in the FOL output.

3.6 Sample running of code

To run the code, the set of sentences must be piped in, and the locations of the CANDC parser binaries must be included in the code (this is specified in detail in the comments). Outputs from a sample run are shown in the figure.

[5] http://stackoverflow.com/questions/1833252/java-stanford-nlp-part-of-speech-labels

```
sriram@GLaDOS:~/builds/natural-lang-to-fol$ python2 krr_assignment.py
Enter sentence:
Alice heard Bob
Enter more? (y/n)
y
The girl sees Bob
Enter more? (y/n)
y
girls see boys
Enter more? (y/n)
y
A boy eats an apple
Enter more? (y/n)
y
The boys watch television
Enter more? (y/n)
y
A girl eats oranges
Enter more? (y/n)
y
Alice eats a banana and an orange
Enter more? (y/n)
y
The boys see a bat or a ball
Enter more? (y/n)
n
Processing...
  heard(Alice,Bob)
There_exists x1  girl(x1) and  sees(x1,Bob)
For_all x1 For_all x2 girl(x1) and boy(x2) --> see(x1, x2)
There_exists x1 There_exists x2  boy(x1) and apple(x2) and  eats(x1,x2)
There_exists x1 television(x1) and For_all x2 boy(x2) --> watch(x2, x1)
There_exists x1 For_all x2  girl(x1) and orange(x2) -->  eats(x1,x2)
There_exists x1 There_exists x2  banana(x1) and orange(x2) and  eats(Alice,x1) and eats(Alice,x2)
There_exists x1 bat(x1) and There_exists x2 ball(x2) For_all x3 boy(x3) --> see(x3, x1) or see(x3, x2)
sriram@GLaDOS:~/builds/natural-lang-to-fol$
```

4 Conclusion

Thus, using a parser, and with a basic understanding of FOL constructs, we have built a program that translates a subset of natural language to FOL; it is not possible to express most complicated sentences (such as, say, this one) in FOL, but we have covered most cases which have a direct translation to the language of quantifiers and predicates. The code has been written in an object-oriented manner in Python, and can easily be extended to tackle more cases as and when required.