Conversion of First Order Logic(FOL) to Clause Form(CNF)

CS6770: Knowledge, Representation and Reasoning

Team 12
Problem Statement 2

Submitted by:
R Rishaab Karthik (MM19B046),
B Aditi (MM19B022),

Contents

Conversion of FOL to Clause Form

1 Introduction

In the given problem, the input is in FOL format and is expected to be converted to Clause form/CNF. The project specifications are as follows:

Input format: .txt file
Input language: First Order Logic (FOL)
Output format: .XML and .txt
Output sub-language: Clausee Normal Form (CNF)
Programming Language: Python

1.1 What is FOL?

First Order Logic is a Formal Language used for knowledge representation in AI. It consists of its own syntax and semantics and is often considered as an extension to propositional logic. However, unlike propositional logic which only deals with simple declarative sentences, FOL is also inclusive of predicates and quantification.

1.2 What is Clause Form(Conjunctive Normal Form)?

Clause form in Boolean Logic is described as a conjunction of one or more clauses, where, a clause is a disjunction of literals. Any formula written in formal logic may be represented in the CNF format by the process of resolving using FOL rules. It can also be called as a "sub-language" in FOL.

Typically, a formula in clause form may be represented as follows:

$$C_1 \wedge C_2 \wedge ...... \wedge C_n$$

Where, $C_1, C_2, ..., C_n$ are clauses. Each clause maybe represented as follows:

$$L_1 \vee L_2 \vee .... \vee L_m$$

Where, $L_1, L_2, ...\ L_m$ are called literals. A literal may be defined as a single clause.

The approach to the given problem statement is explained in the next section.

2 Approach and Algorithm

In order to convert the FOL input to CNF, we have to represent all connectives present in the functions using only AND, OR and NOT operators. This conversion is primarily done as implementation of resolution refutation method requires the input to be in CNF format. In order for the replacement of operators to occur, FOL rules are implemented (which are enlisted further in this section).

The first step in simplifying the an FOL statement to CNF form is to remove all the implications,double implications and implied by statements into operators with or and not. For that we use the following steps in order:

1. $A \Leftrightarrow B$ is replaced by $A \Rightarrow B$ and $A \Leftarrow B$
2. $A \Leftarrow B$ is replaced by $B \Rightarrow A$
3. $A \Rightarrow B$ is replaced as $\neg A \vee B$

The next step now is to simplifying the expression further by moving any negation operator in front of binary operators/ quantifiers like forall,.exists,and , or. The below steps are followed here ,

1. The negation of a universal quantifier results in the existential quantifier and similarly, the negation of the existential quantifier is the universal quantifier. $\neg\forall X_1,..X_n\ A(X_1,..X_n) \equiv \exists X_1,..X_n\ \neg A(X_1,..X_n)$ and $\neg\exists X_1,..X_n\ A(X_1,..X_n) \equiv \forall X_1,..X_n\ \neg A(X_1,..X_n)$
2. The negation in front of the $\wedge$ operator gives negation of the both the operands , along with $\wedge$ being changed to $\vee$.
   $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$
3. Similarly in the case of the $\vee$ operator it leads to negation of both the operands , and the $\vee$ operator changes to $\wedge$.
   $\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$
4. The negation of a negated predicate simplifies to the predicate itself.
   $\neg(\neg A) \equiv A$

After the above steps now it is to be ensured that the names of all variables are standardised and unique on each sentence, for this we subscript the variable names with numbers. Skolemization of variables: We drop the existential quantifier of variables by following the following steps:

1. An existentially quantified variable not in the scope of a universal quantifier is replaced with a skolem constant
2. An existentially quantified variable within the scope of another universal quantifier is replaced with a skolem function of the universally quantified variables.

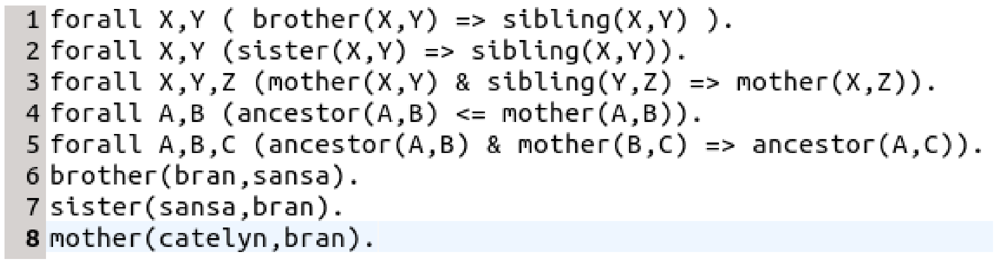
The final step is to drop all the universal quantifiers as well as we have replaced all existential variables by skolemization. We also made a custom recursive parser on python itself to convert the output in xml to txt format. The code for which is given below:

```
def Parsers(j):                                                    #parser to change
    if(j.tag=="PREDICATE"):                                     #xml to txt form
        li=[]
        for k in j.getchildren():
                li.append(k.text)
        return j.get('name')+'('+','.join(li)+')'

    if(j.tag=="NOT"):
        currString=Parsers(j.getchildren()[0])
        return "~("+currString+")"

    if(j.tag=="AND"):
        currString1=Parsers(j.getchildren()[0])
        currString2=Parsers(j.getchildren()[1])
        return '('+currString1+") and ("+currString2+")"

    if(j.tag =="OR"):
        currstring3=Parsers(j.getchildren()[0])
        currString4=Parsers(j.getchildren()[1])
        return "(" + currstring3+ ") or ("+currString4+")"
```

3 Sample I/O

3.1 Input

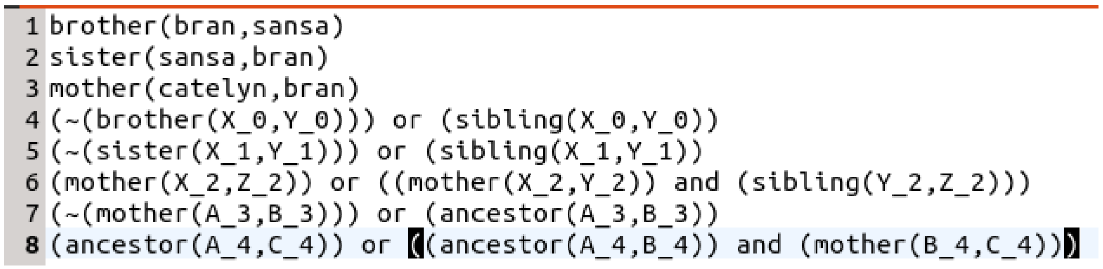
Contents of input text file :]



```
forall X,Y ( brother(X,Y) => sibling(X,Y) ).
forall X,Y (sister(X,Y) => sibling(X,Y)).
forall X,Y,Z (mother(X,Y) & sibling(Y,Z) => mother(X,Z)).
forall A,B (ancestor(A,B) <= mother(A,B)).
forall A,B,C (ancestor(A,B) & mother(B,C) => ancestor(A,C)).
brother(bran,sansa).
sister(sansa,bran).
mother(catelyn,bran).
```

3.2 Output

The corresponding output text file in cnf form:



```
brother(bran,sansa)
sister(sansa,bran)
mother(catelyn,bran)
(~(brother(X_0,Y_0))) or (sibling(X_0,Y_0))
(~(sister(X_1,Y_1))) or (sibling(X_1,Y_1))
(mother(X_2,Z_2)) or ((mother(X_2,Y_2)) and (sibling(Y_2,Z_2)))
(~(mother(A_3,B_3))) or (ancestor(A_3,B_3))
(ancestor(A_4,C_4)) or ((ancestor(A_4,B_4)) and (mother(B_4,C_4)))
```

4 Readme

The project has an input and output folder added to the Krr parser files given The main code is given in the foltocnf.py file and it can be run from the command line

```
~$:python foltocnf.py
```

The input folder should contain the textfile which has all the fol statements in it. The name of the text file is stored as 1.txt by default in the variable inputtxt, which can be modified for any other input file. The input folder also contains the xml parsed version of the same text file when the code is run.

The output folder contains both the output xml file and the output txt file