CS6770
KRR Programming Assignment

Problem statement :

Group D

8. ALC Taxonomy Builder - Subsumption Hierarchy

Team Members: Rudrik Shah(CS22M073)
Amishi Panwar (CS22M009)

Introduction

Logic-based knowledge representations implement mathematical logic, a subfield of mathematics dealing with formal expressions, reasoning, and formal proof.

Description logics (DL) constitute a family of formal knowledge representation languages that provide a logical underpinning for OWL ontologies. Many description logics are more expressive than propositional logic, which deals with declarative propositions and does not use quantifiers, and more efficient in decision problems than first-order predicate logic, which uses predicates and quantified variables over nonlogical objects.

A crucial design principle in description logic is to establish favorable trade-offs between expressivity and computational complexity to suit different applications. The expressivity of each description logic is determined by the supported mathematical constructors. There are crisp general-purpose DLs, spatial, temporal, and spatiotemporal DLs, probabilistic and possibilistic DLs, and fuzzy DLs

ALC :

ALC stands for Attributive Language with Complements. An ALC KB contains two parts:

- Define terminology: TBox
- Give assertions: ABox

Main components of the TBox:

- Concepts: Represent classes of individuals
- Roles: Represent binary relations between individuals
- Complex concepts using constructors

Examples:

- Concept names: Person, Female
- Role names: ParentOf, HasHusband
- Individual names (in the ABox): John, Mary

Subsumption Hierarchy:

The subsumption algorithm determines subconcept-superconcept relationships: a concept C is subsumed by a concept D w.r.t. a TBox if, in each model of the TBox, each instance of C is also an instance of D. Such an algorithm can be used to compute the taxonomy of a TBox, i.e., the subsumption hierarchy of all those concepts introduced in the TBox.

Negated Normal Form:

In Negated Normal Form, a formula has only $\sqcap$, $\sqcup$ and $\neg$ connectives and negation occurs only before concept names. We have to push negation down to sub-formulas until they only occur before concept names.

Problem statement

We are given a text file in which the concept descriptions are there. Those concept descriptions are represented using subclass-parent class relation, equivalence relation, disjoint union relation etc. Below is a snapshot of the input text file.

```
#P8 - Taxonomy Extraction:
#Concept                SubClassOf
#---------------------------------------------------
#Person
#Course          ~Person
#UGC             Course
#PGC             Course
#Teacher                 Person, ∃teaches.Course
#Student                 Person, ∃attends.Course

Class: Person

Class: Course
        SubClassOf:     not Person

Class: UGC
        SubClassOf:     Course

Class: PGC
        SubClassOf:     Course

Class: Teacher
        EquivalentTo:   Person and teaches some Course

ObjectProperty: teaches
        Domain:                 Person

Class: Student
        EquivalentTo:   Person and attends some Course

ObjectProperty: attends
        Domain:                 Person

Individual: Mary
        Types:          Person
        Facts:          teaches CS600, attends Ph456

Individual: CS600
       Types:         Course

Individual: Ph456
       Types:         Course, PGC

Individual: Hugo
       Types:         Person
       Facts:         teaches Ph456

Individual: Betty
       Types:         Teacher
       Facts:         attends Ph456
```

We are also given an XML parser which converts this file to an XML file.

Implementation

1. **Red input file** : Used **xml.etree.ElementTree** library to read the xml file and return the tree value.

2. Create a **dictionary** which will have class names as **keys** and the super classes of the corresponding classes as the **values** for that particular key.
   Dictionary = {key : value} = {class : set of super classes of this class}

3. Iterate all the class tag members in the input file and get the child of each **“Class”** tag.

```
for child in root.findall("Class"):
```

   A “Class” tag can have three kinds of child tags - “CONCEPT”, “SubClassOf” and “EquivalentTo”

4. The “CONCEPT” tag, which is an immediate child of the “Class” tag, will give the name of the class.

```
# Get class name as key
key = child.find('CONCEPT').text

# Get subclasses in a set
setData = set()
```

   Extract the name of the class and add it to the dictionary as a key and set its value to empty.

5. The “SubClassOf” tag contains the super classes of the current class.

We have to list all the classes inside this tag by getting the “CONCEPT” tags inside this and add them to the set.

Further, a “SubClassOf” tag can have “NOT”, “EXISTS”, “AND” tags which can be further traversed down to get the “CONCEPTS” tag within them and add these concepts with required keywords to the set.

```
    # Handle "SubClassOf" part
    for newChild in child.findall('SubClassOf'):

        #handle the EXISTS case
        if(newChild.find('EXISTS') != None):
            setData.add(newChild.find('ROLE').text +
                        " some " + newChild.find('CONCEPT').text)

        # Handle the Not subclass case
        if(newChild.find('NOT') != None):
            for notChild in newChild.findall('NOT'):
                setData.add("not "+notChild.find('CONCEPT').text)

        # Get all sub class values
        if(newChild.find('CONCEPT') != None):
            setData.add(newChild.find('CONCEPT').text)

        # Handle the AND subclass case
        if(newChild.find('AND') != None):

            for newChild in child.findall('EquivalentTo'):

                if(newChild.find('NOT') != None):
                    for notChild in newChild.findall('NOT'):
                        setData.add("not "+notChild.find('CONCEPT').text)

                if(newChild.find('EXISTS') != None):
                    setData.add(newChild.find('ROLE').text +
                                " some " + newChild.find('CONCEPT').text)

                if(newChild.find('CONCEPT') != None):
                    setData.add(newChild.find('CONCEPT').text)
```

6. The “EquivalentTo” tag contains the equivalent classes of the current class.
   We have to list all the classes inside this tag by getting the “CONCEPT” tags inside this and add them to the set.
   Further, a “EquivalentTo” tag can have “NOT”, “EXISTS”, “AND” tags which can be further traversed down to get the “CONCEPTS” tag within them and add these concepts with required keywords to the set.

```
# Handle "Equivalent to" part
for newChild in child.findall('EquivalentTo'):

    if(newChild.find('NOT') != None):
        for notChild in newChild.findall('NOT'):
            setData.add("not "+notChild.find('CONCEPT').text)

    if(newChild.find('EXISTS') != None):
        setData.add(newChild.find('ROLE').text +
                    " some " + newChild.find('CONCEPT').text)

    if(newChild.find('CONCEPT') != None):
      setData.add(newChild.find('CONCEPT').text)

    if(newChild.find('AND') != None):
        for newnewChild in newChild.findall('AND'):

            if(newnewChild.find('NOT') != None):
                for notChild in newnewChild.findall('NOT'):
                    setData.add("not "+notChild.find('CONCEPT').text)

            if(newnewChild.find('EXISTS') != None):
                setData.add(newnewChild.find('EXISTS').find(
                    'ROLE').text+" some " + newnewChild.find('EXISTS').find('C

            if(newnewChild.find('CONCEPT') != None):
                setData.add(newnewChild.find('CONCEPT').text)

dictionary[key] = setData
```

7. Assign the set used above to the respective key/“Class”.

8. Handling the transitivity : This part of the code handles the transitivity part of the subsumption hierarchy.
   This can be done iteratively by adding the superclass of the super class of the current class to the set in the dictionary where the key is the current class and the value is the set of the current class.

```
# Handle the transitivity
dic = {}
while(dic != dictionary):
    dic = dictionary.copy()
    for key in dictionary:
        news = dictionary[key]
        for item in dictionary[key]:
            if item in dictionary:
                for i in dictionary[item]:
                    if i in dictionary:
                        news = news.union({i

        dictionary[key] = news
```

9. **Write to the output file :** Write the data to the output file.

```
# write the data to the ouput file
file = open("output/output.txt", 'w')

for data in dictionary:
    file.write('Class: ' + data + '\n')
    listData = list(dictionary[data])

    if bool(dictionary[data]):
        file.write("\tSubClassOf: ")
        for item in listData[:-1]:
            file.write(item+', ')

        file.write(listData[-1])
        file.write("\n")
```

Code structure :

Code : This folder contains the main implementation of assignment. Code will take the ALC KB in Negation Normal Form (xml format) which generates the build the taxonomy (subsumption hierarchy) of concepts defined in the KB. Output file is exported to **output/output.txt** which is used to create the output in txt file format.

Input : Contains the ALC KB in Negation Normal Form (xml files)

Parser : KRR software package which converts the text(output from assignment implementation) file to xml format.

Output

```
<KB>
  <Class>
    <CONCEPT>Person</CONCEPT>
  </Class>
  <Class>
    <CONCEPT>Course</CONCEPT>
    <SubClassOf>
      <NOT>
        <CONCEPT>Person</CONCEPT>
      </NOT>
    </SubClassOf>
  </Class>
  <Class>
    <CONCEPT>UGC</CONCEPT>
    <SubClassOf>
      <CONCEPT>Course</CONCEPT>
    </SubClassOf>
  </Class>
  <Class>
    <CONCEPT>PGC</CONCEPT>
    <SubClassOf>
      <CONCEPT>Course</CONCEPT>
    </SubClassOf>
  </Class>
  <Class>
    <CONCEPT>Teacher</CONCEPT>
    <SubClassOf>
      <CONCEPT>Person</CONCEPT>
      <EXISTS>
        <ROLE>teaches</ROLE>
        <CONCEPT>Course</CONCEPT>
      </EXISTS>
    </SubClassOf>
  </Class>
  <Class>
    <CONCEPT>Student</CONCEPT>
    <SubClassOf>
      <CONCEPT>Person</CONCEPT>
      <EXISTS>
        <ROLE>attends</ROLE>
        <CONCEPT>Course</CONCEPT>
      </EXISTS>
    </SubClassOf>
  </Class>
</KB>
```