VIET NAM NATIONAL UNIVERSITY,HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF: INFORMATIONTECHNOLOGY

Project 02

Introduction to Artificial Intelligence

Name: Lai Minh Thong
Student ID: 20127635

Instructors:

Dr. Bui Tien Len

Dr. Nguyen Ngoc Duc

Ho Chi Minh City – 2022

Project 02 – Problem 4: Inference

I. Completeness.

Index	Description	Ratio
1	Read input data and store in suitable data structure	100%
2	Implementation of resolution method	100%
3	Inference process and results	100%
4	Testcases, report, evaluations	100%

II. Detail Implementation.

Source: https://github.com/ThongLai/HCMUS_LinearRegressionProject

The reference is noted by a subscript number referring to the links. Aggregation of links is put in the *References* section.

The formulas and pseudo code are taken from the slide of the learning materials. Implementation is referenced at [1] https://github.com/t3bol90/PL-Resolution

[2] https://github.com/IceIce1ce/PL-Resolution-HCMUS

[3] [2]

Quick start:

Propositional Resolution - Inference Rule takes two premises in the form of clauses (A ∨ x) and (B ∨ ¬x) and gives the clause (A ∨ B) as a conclusion. The two premises are said to be resolved and the variable x is said to be resolved away. Resolving the two clauses x and x gives the empty clause. [3]

Resolution inference rule (for CNF):

$$\frac{\ell_1 \vee \cdots \vee \ell_i \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_j \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n} \quad (8)$$

where ℓ_i and m_j are complementary literals

Theorem 3

Resolution inference rule is sound and complete for CNF KB

Algorithm:

Pseudo-code for the function:

The resolution algorithm

- Proof by contradiction: To show that $KB \models \alpha$, prove that $KB \wedge \neg\alpha$ is unsatisfiable

```
function PL-Resolution(KB, α) returns true or false
inputs: KB, the knowledge base, a sentence in propositional logic
        α, the query, a sentence in propositional logic
  clauses ← the set of CNF clauses of KB ∧ ¬α
  new ← ∅
  loop do
    for each pair of clauses Ci, Cj in clauses do
      resolvents ← PL-Resolve(Ci, Cj)
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Implement:

```
# Propositional logic resolution inference rule
def PL_Resolution(KB, alpha):
    """ Negate the alpha clauses to add it into KB """
    not_alpha = negate(alpha)

    """ Add KB AND NOT alpha into KB """
    clauses = (KB + [not_alpha])

    generatedData = []
    while True:
        new_clauses = []
        can_entail = False

        clauses_pairs = [(clauses[i], clauses[j]) for i in range(len(clauses))
for j in range(i+1,len(clauses))]

        for pair in clauses_pairs:
            resolvents = PL_Resolve(*pair)

            if resolvents == False:
                continue
            elif len(resolvents) == 0:
                can_entail = True
                new_clauses.append(['{}'])
                break
            else:
                if resolvents not in (clauses + new_clauses):
                    new_clauses.append(resolvents)

        generatedData.append([str(len(new_clauses))])
        generatedData.extend(new_clauses)

        if can_entail:
            generatedData.append(['YES'])
            return can_entail, generatedData
        elif len(new_clauses) == 0:
            generatedData.append(['NO'])
            return can_entail, generatedData

        clauses.extend(new_clauses)
```

My implementation of the function has a slight difference from the pseudo-code algorithm. In particular:

- The *generatedData* list is for the purpose of writing data to output files which later being used as a parameter of the *writeFile()*. This list contains the newly generated clauses in the required form for this problem. The PL_Resolution will return this and the boolean variable indicates the entailable aspect of the resolution.
- Instead of returning the results of the resolution process in the for loop through each pair of clauses. My implement has a boolean *can_entail* to determine whether it is time to stop the process and move to the final steps corresponding to the value of *can_entail* (Append the last statement “YES”/”NO” to the *generatedData* or

update the KB’s clauses)

Test cases:

input1.txt:

```
-A
4
-A OR B
B OR -C
A OR -B OR C
-B
```

output1.txt:

3

-A

B

-C

4

-B OR C

A OR C

A OR -B

{}

YES

This test the case of KB can entailed alpha which is *'-A'*.

input2.txt:

A

4

-A OR B

-C OR B

A OR C OR -B

-B

output2.txt:

2

-C

-B OR C

2

-A OR C

A OR -B

1

A OR -C

1

B OR -C

0

NO

This test the case of KB can not entailed alpha which is *‘A’*.

III. References

Project 02 – Propositional Logic Resolution.

https://github.com/ThongLai/HCMUS_Propositional-logic-resolution

References.

[1] https://github.com/t3bol90/PL-Resolution

[2] https://github.com/IceIce1ce/PL-Resolution-HCMUS

[3] https://www.sciencedirect.com/topics/computer-science/resolution-inference