A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

16/04/2022

# REPORT

Logic Inference Project

Fundamentals of Artificial Intelligence

Several thin, curved lines in dark blue and light grey originate from the left side and curve upwards and to the right.

## Instructors

Pham Trong Nghia

Nguyen Thai Vu

**BUI QUOC HUNG**

VNUHCM – UNIVERSITY OF SCIENCE

## Mục lục

Introduction .....	2
About the author .....	2
Problem statement .....	2
Implementation .....	3
Experiments .....	4
Conclusion.....	14
Pros .....	14
Cons.....	14
Potential Improvements .....	14
Evaluation .....	14
References .....	15

## Introduction

### About the author

Student name: Bui Quoc Hung

Student ID: 20127508

### Problem statement

Given a knowledge base (KB) with  $\alpha$  and both of them are represented by propositional logic and standized into CNF. Determine KB entails  $\alpha$  ( $KB \models \alpha$ ) by resolutions.

## Implementation

First, according to the problem requirements, the input are stored in input.txt, therefore we use built-in modules of Python to read line-by-line. For each statement, we preprocess them, convert them into string, then store them into several variable to keep their information.

Once we done for the preparation stage, we call PL\_Resolution to start our resolution progress. In general, the main idea of the algorithms is almost represented by fig1.

---

Figure 7.13

---

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
  inputs:  $KB$ , the knowledge base, a sentence in propositional logic
            $\alpha$ , the query, a sentence in propositional logic

   $clauses \leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$ 
   $new \leftarrow \{ \}$ 
  while true do
    for each pair of clauses  $C_i, C_j$  in  $clauses$  do
       $resolvents \leftarrow$  PL-RESOLVE( $C_i, C_j$ )
      if  $resolvents$  contains the empty clause then return true
       $new \leftarrow new \cup resolvents$ 
    if  $new \subseteq clauses$  then return false
   $clauses \leftarrow clauses \cup new$ 
```

A simple resolution algorithm for propositional logic. PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

---

*Fig1: Propositional logic resolution algorithm – via Artificial Intelligence: A Modern Approach*

First, we push  $\neg\alpha$  into the clauses set (this source code does not support  $\alpha$  with multiple literals inside), then according to the requirements “All the sentences  $A \vee B \vee \neg B$  have the truth value is True, then we should eliminate it”, we call a redundant-filtering function to eliminate any clauses like that. After that, we consider all pair of clause in the set of clauses that we have processed, called PL\_Resolve, which performs resolution if there is any pair of negative and non-negative literal in that pair, then the function returns the results.

- If the resolvents is empty, then the function PL\_Resolution returns true, which means we can performs the query *alpha* by the given KB

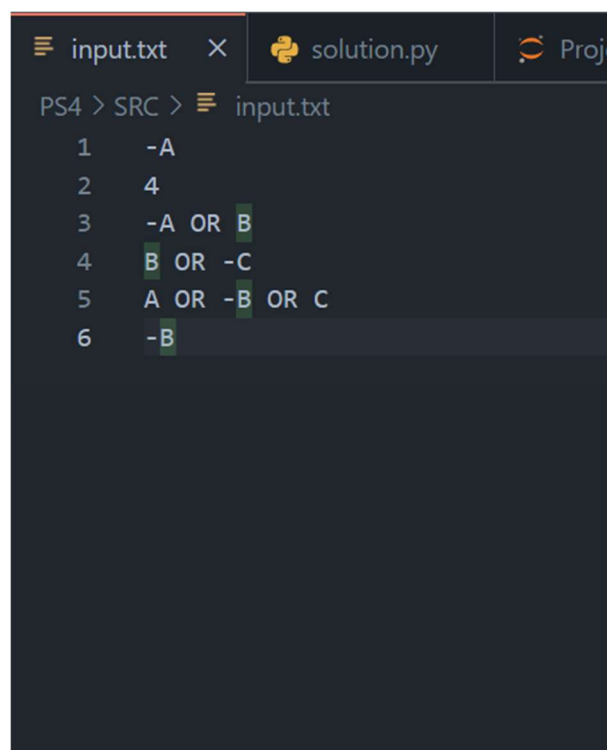
After each time we invoke PL\_Resolve, we store the resolvents into a set, called “new”. If “new” is the subset of clauses, which indicates we can’t performs *alpha* from the KB, return False. If it not, merge the new storage into the set of clauses then continue to resolve these clauses.

After the PL\_Resolution function reaches its terminal state, following the format in the problem file, we write the logs storage – which save the resolvents if it does not appears in clauses storage – into the output.txt by the number of resolvents and resolvents in each loop. Finally, we display the terminal state of PL\_Resolution at the end of the file, YES for true and NO for false.

## Experiments

In the source code, I also display the 2 reason factors of each resolvent, so we can easily understand of the algorithm. According to the requirements, the algorithm does not stop when the resolvents so there are also inference even the resolvents is empty. Because I have used set – built-in data structure in my source code – the order of clauses after being pushed into set is messy since they use several kinds of stuff random stuff with unknown seed so I can't control the order. However, my implementation is reliable, which is demonstrated in these following examples.

First example (from the problem statement)



```
input.txt x solution.py Project
PS4 > SRC > input.txt
1 -A
2 4
3 -A OR B
4 B OR -C
5 A OR -B OR C
6 -B
```

Fig2: Sample 1's input

The screenshot shows a Jupyter Notebook with four tabs: `input.txt`, `solution.py`, `Project02_logic.ipynb`, and `output.txt`. The `output.txt` tab is active, displaying a list of 10 items:

- 1 3
- 2 -C
- 3 -A
- 4 B
- 5 4
- 6 {}
- 7 A OR -B
- 8 -B OR C
- 9 A OR C
- 10 YES

Below the list, the `TERMINAL` tab is active, showing the output of running `python main.py` in a Windows PowerShell environment. The output includes a copyright notice and a list of results for various logical expressions:

```
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'-C'}, results from B;-C and -B
{'-A'}, results from -A;B and -B
{'B'}, results from -A;B and A
{''}, results from -A and A
{'A;-B'}, results from A;-B;C and -C
{'-B;C'}, results from A;-B;C and -A
{'A;C'}, results from A;-B;C and B
[3, ['-C', '-A', 'B'], 4, ['', 'A;-B', '-B;C', 'A;C']]
PS D:\repo\logic-inference-ai\PS4\SRC>
```

Fig3: Sample 1's output

Second example (from the problem statement)

```
input.txt M ×  output.txt M  solution.p
PS4 > SRC > input.txt
1  A
2  4
3  -A OR B
4  -C OR B
5  A OR C OR -B
6  -B
```

*Fig4: Sample 2's input*

```
input.txt M  solution.py  Project02_logic.ipynb  output.txt M X

PS4 > SRC > output.txt
1  2
2  -C
3  C OR -B
4  1
5  A OR -B
6  0
7  NO

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'-C'}, results from B;-C and -B
{'-A'}, results from -A;B and -B
{'B'}, results from -A;B and A
{''}, results from -A and A
{'A;-B'}, results from A;-B;C and -C
{'-B;C'}, results from A;-B;C and -A
{'A;C'}, results from A;-B;C and B
[3, ['-C', '-A', 'B'], 4, ['', 'A;-B', '-B;C', 'A;C']]
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'-C'}, results from -C;B and -B
{'C;-B'}, results from A;C;-B and -A
{'A;-B'}, results from A;C;-B and -C
[2, ['-C', 'C;-B'], 1, ['A;-B'], 0, []]
PS D:\repo\logic-inference-ai\PS4\SRC> 
```

Fig5: Sample 2's output

Third example



```
input.txt M X output.txt M solu
PS4 > SRC > input.txt
1 A
2 7
3 -A OR B OR D OR E
4 -C OR B
5 A OR -B OR C OR -E
6 -B OR C
7 A OR E
8 -D
9 -E
```

*Fig6: Sample 3's input*

☒ input.txt M
☒ output.txt M X
☒ solution.py
☒ Project02\_logic.ipynb
☒ main.py M

PS4 > SRC > ☒ output.txt

```

1 5
2 -A OR B OR E
3 -A OR B OR D
4 E
5 A
6 -B OR C OR -E
7 6
8 B OR D OR E
9 B OR D
10 -A OR B
11 B OR E
12 {}
13 A OR -B OR C
14 YES

```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER

```

[3, ['-A;B;E', 'E', '-B;C;-E'], 1, ['A;-B;C'], 0, []]
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'-A;B;E'} , results from -A;B;D;E and -D
{'-A;B;D'} , results from -A;B;D;E and -E
{'E'} , results from A;E and -A
{'A'} , results from A;E and -E
{'-B;C;-E'} , results from A;-B;C;-E and -A
{'B;D;E'} , results from -A;B;D;E and A
{'B;D'} , results from -A;B;D and A
{'-A;B'} , results from -A;B;D and -D
{'B;E'} , results from -A;B;E and A
{''} , results from -E and E
{'A;-B;C'} , results from A;-B;C;-E and E
[5, ['-A;B;E', '-A;B;D', 'E', 'A', '-B;C;-E'], 6, ['B;D;E', 'B;D', '-A;B', 'B;E', '', 'A;-B;C']]

```

Fig7: Sample 3's output

Fourth example

```
input.txt M ×  output.txt M  solution.py  Proj
PS4 > SRC > input.txt
1  -F
2  10
3  -F
4  A OR -B OR C OR -E
5  -B OR C
6  A OR E
7  -A OR B OR D OR E
8  -B OR C
9  -D
10 -E
11 G
12 D OR -E OR G
```

*Fig8: Sample 4's input*

☒ input.txt M
☒ output.txt M X
☒ solution.py
☒ Project02\_logic.ipynb
☒ main.py M

PS4 > SRC > ☒ output.txt

```

1 5
2 A
3 -A OR B OR D
4 -A OR B OR E
5 {}
6 -E OR G
7 YES

```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER

```

{'B;D;E'} , results from -A;B;D;E and A
{'B;D'} , results from -A;B;D and A
{'-A;B'} , results from -A;B;D and -D
{'B;E'} , results from -A;B;E and A
{' ' } , results from -E and E
{'A;-B;C'} , results from A;-B;C;-E and E
[5, ['-A;B;E', '-A;B;D', 'E', 'A', '-B;C;-E'], 6, ['B;D;E', 'B;D', '-A;B', 'B;E', ' ', 'A;-B;C']]
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'A'} , results from A;E and -E
{'-A;B;D'} , results from -A;B;D;E and -E
{'-A;B;E'} , results from -A;B;D;E and -D
{' ' } , results from -F and F
{'-E;G'} , results from D;-E;G and -D
[5, ['A', '-A;B;D', '-A;B;E', ' ', '-E;G']]
PS D:\repo\logic-inference-ai\PS4\SRC>

```

Fig9: Sample 4's output

Fifth example

```
input.txt M X output.txt M solutio
PS4 > SRC > input.txt
1 -E
2 12
3 -A OR B OR D OR E
4 -B OR C
5 -D
6 G
7 -F
8 A OR -B OR C OR -E
9 -B OR C
10 A OR E
11 B OR H OR R
12 D OR -E OR G
13 -P OR Q OR Z
14 U OR V OR -S
```

Fig10: Sample 5's input

```
input.txt M  output.txt M X  solution.py  Project02_logic.ipynb  main.py M

PS4 > SRC > output.txt
1  4
2  -A OR B OR E
3  -E OR G
4  D OR G
5  A OR -B OR d
6  0
7  NO

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

{'A'} , results from A;E and -E
[7, ['A;-B;C', '', '-A;B;D', '-A;B;E', 'D;G', '-E;G', 'A']]
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'A;-B;C'} , results from A;-B;C;-E and E
{'-A;B;E'} , results from -A;B;D;E and -D
{'-E;G'} , results from D;-E;G and -D
{'D;G'} , results from D;-E;G and E
[4, ['A;-B;C', '-A;B;E', '-E;G', 'D;G'], 0, []]
PS D:\repo\logic-inference-ai\PS4\SRC> python main.py
{'-A;B;E'} , results from -A;B;D;E and -D
{'-E;G'} , results from D;-E;G and -D
{'D;G'} , results from D;-E;G and E
{'A;-B;C'} , results from A;-B;C;-E and E
[4, ['-A;B;E', '-E;G', 'D;G', 'A;-B;C'], 0, []]
PS D:\repo\logic-inference-ai\PS4\SRC>
```

Fig11: Sample 5's output

## Conclusion

### Pros

- If the knowledge base is enough, the algorithm can find the suitable solution.
- The algorithm is simple and easy to understand and implementation

### Cons

- There are so many redundant steps that this algorithm performs if the KB's clauses set are not focus on the query
- In general, propositional logic has limited expressive power, for example, all, some,... so the algorithm need various clauses and literals to represent the problem and solve it.
- Takes much time in the resolution function since we need to consider all pair of clauses and all pair of literals within them.
- It is hard to extend the KB to solve a very different query compare to the original KB
- Inference takes time since it must iterates all the KB's clauses
- Difficult to represent KB If the KB is changed overtime

### Potential Improvements

- According to the problem, the input follows CNF, so there are not any kind of inputs  $C1$ ,  $C2$  that need to be resolved, between both complex clauses, for example:  $A \vee B \vee C$  and  $\neg(B \vee C)$ , and  $\neg(B \vee C) = \neg B \wedge \neg C$ , and there is not any clause like that in CNF standard, so just let  $C1$  is always longer than  $C2$  (length of literal of  $C2$  is always 1, as I mentioned above), then just construct each longer clauses into a binary search tree (Red Black Tree, AVL Tree, etc.), then search its negative literal efficiently.  
⇒ Faster inference progress

### Evaluation

No.	Criteria	Percentage of completeness
1	Read the input and store those data in suitable data structure	100%
2	Implement resolution for propositional logic	100%
3	Each step of inference progress generate completely clauses and correct conclusion	100%
4	Follow the description of format	100%
5	Report contains test case and evaluation	100%

## References

[1] Artificial Intelligence: A Modern Approach, Fourth Edition, Chapter 7