REPORT FOR LAB 02-PROPOSITIONAL LOGIC RESOLUTION AND APPLICATIONS

TEACHER: PHD. NGUYEN NGOC THAO, MSC. HO THI THANH TUYEN

COURSE: INTRODUCTION OF ARTIFICIAL INTELLIGENCE - CSC14003 (18CLC2)

STUDENT: VO TRAN QUANG TUAN – ID: 18127248

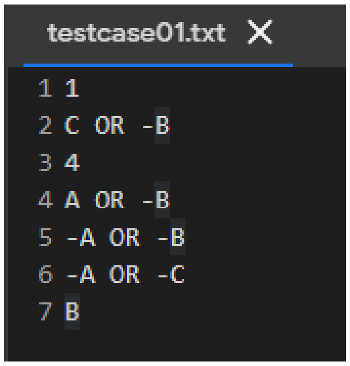
HO CHI MINH UNIVERSITY OF SCIENCES – FACULTY OF INFORMATION TECHNOLOGY

PROBLEM DESCRIPTION

- Given a knowledge base (KB) and a query α, both are sets of propositional clauses in CNF.
- Implementing the PL-RESOLUTION function to check whether KB entails α (KB $\vDash \alpha$).
- Provide 5 non-trivial test cases, along with the submission, to validate the program.
- Discussing the PL-resolution algorithm's efficiency and suggest some solution(s) to address the limitations.

INPUT SPECIFICATIONS

- The input file stores the KB and α, whose format is as follows:
  - The first line contains a positive integer M, which is the number of clauses in query α.
  - M next lines represent the clauses in query alpha, one clause per line.
  - The M+2 line contains a positive integer N, which is the number of clauses in the KB.
  - N next lines represent the clauses in KB, one clause per line.
- A (positive) literal is represented by one uppercase letter, i.e. A-Z. A negated literal is a literal associated with a minus symbol (-) right ahead. Literals are connected by the OR keyword.
- There may be multiple white spaces between literals and keywords.
- Example for the input file (Figure 1):
  - The first line: 1 -> one clause need to be queried by the KB.
  - The 2nd line: C OR -B: clause need to be queried.
  - The 3rd line: 4 -> four clauses in KB (four axioms).
  - The four next lines: A OR -B, -A OR -B, -A OR -C, B -> they are four axioms in KB.

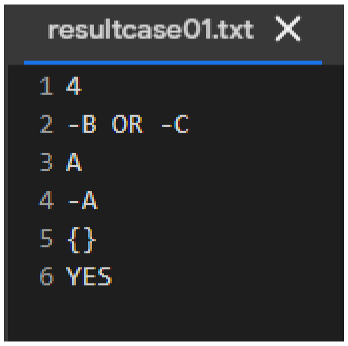


```
1
C OR -B
4
A OR -B
-A OR -B
-A OR -C
B
```

Figure 1: Example for the input file.

OUTPUT SPECIFICATIONS

- The output file stores the set of clauses generated during the resolution and the conclusion.
- The output format is as follows:
    - The first line contains a non-negative integer M1, which indicates the number of clauses generated in the first loop.
    - M1 next lines represent the newly generated clauses in the first loop (including the empty clause). An empty clause is represented by the string "{}".
    - Subsequent loops (of M2, M3, …, Mn clauses) are represented as stated above.
    - The last line shows the conclusion, i.e. whether "KB entails α". Print YES if KB entails α. Otherwise, print NO.
    - Duplicates, i.e., clauses that are identical to some clauses appeared previously (e.g., in the current/previous loop or in the original KB) are ignored.
- Example for the output file (Figure 2):
    - The 1st line: 4 -> fours clauses were generated by resolving.
    - The final line: YES -> KB can entail α.



```
4
-B OR -C
A
-A
{}
YES
```

Figure 2: Example for the output file.

CODE SPECIFICATIONS

SOURCE CODE FILE FORMAT

- In this lab, I use file jupyter notebook (.ipynb) and python file (.py) to code, they are pl_resolution.ipynb and pl_resolution.py. You can choose one of them to check the code, or all of them.
- If you want to test the program:
    - By using python file (.py), you must use this command in command prompt (window) or terminal (linux, MacOS): `python pl_resolution.py`
    - By using jupyter notebook file (.ipynb), you must use the jupyter notebook, jupyter lab, google Colab or Visual Studio code (have anaconda) to run cell by cell.
    - Note: you must put all file input in current path which you are working to run.

IMPORTANT FUNCTIONS DESCRIPTIONS

- Import some necessary libraries for task: re package (syntax analysis for string).
- Reading the input file and preprocessing data.
  1. Parsing the pure sentence.
     - Getting a sentence in propositional logic OR (in CNF form).
     - Return the list of literals.

       ➔ Function: `parse_PL_sentences(sentence)`
  2. Reading the input file.
     - Getting the path of input file.
     - Return list of sentences in KB which have been parsed, list of sentences needes to be queries which have been parsed.

       ➔ Function: `read_input(path)`
- Finding the contradiction Literals in two sentences.
  - Getting two sentences need to find some contradiction literals.
  - Return list of contradiction literals

    ➔ Function: `find_contrast(lst_literal_01, lst_literal_02)`
- Implement function resolution for a pair of sentences.
  - Getting a pair of sentences need to be resolved.
  - Return the resolved sentence, if fail return [ ], success return difference [ ].

    ➔ Function: `resolve(literals_01, literals_02)`
- Implement the negative form for list of sentences need to be queried.
  - Getting the list of sentences need to be queried which were parsed.
  - Return the list of negative literals in these sentences.

    ➔ Function: `negative_queries(queries)`
- Implement PL-RESOLUTION algorithm.
  - Getting list of sentence in KB which have been parsed, list of sentences need to be queries which have been parsed
  - Return:
    - A list of list of sentence generated in resolution processing.
    - YES if KB entail query, NO if KB can't entail query.

    ➔ Function: `PL_Resolution(KB, queries)`
- Implement the function using for combination a sentence with propositional logic OR (in CNF) from a list of literals.
  - Getting a list of literals.
  - Return a sentence with propositional logic OR form (in CNF).

    ➔ Function: `combination(lst_literals)`
- Implement function using for write data for output file.
  - Getting the input file path, the output file path.
  - Write data in resolution processing to file output.

➔ Function: `write_output(input_path, output_path)`

- Running the PL-RESOLUTION testing.
    - with 5 input files: testcase01.txt -> testcase05.txt.
    - Outputs are 5 files: resultcase01.txt -> resultcase05.txt

➔ Function:

```
write_output('testcase01.txt', 'resultcase01.txt')
write_output('testcase02.txt', 'resultcase02.txt')
write_output('testcase03.txt', 'resultcase03.txt')
write_output('testcase04.txt', 'resultcase04.txt')
write_output('testcase05.txt', 'resultcase05.txt')
```

MORE DISCUSSIONS

- PL-RESOLUTION is a powerful algorithm for inference from propositional logic. It is complete, and can inference all thing from KB by using PL-resolution. So that, is is possible to build a theorem prover that is sound and complete.
- Because we can generate more propositional clauses from this rule, it may be over the bigO of exponent.
- In some case, the strong of PL-resolution is not necessary, because it generate a lot of proposition clauses which are not important for the inference.
- Moreover, in our world, some problem can't be converted to CNF form of propositional clause, so the PL-resolution is impossible.
- In order to address the limit of PL-resolution, there are a lot of algorithms were invented: forward chaining (FC), backward chaining (BC),…
- With forward chaining, we come from KB to infer the query, with backward chaining, we come from query and backprop to the axiom in KB. If using FC or BC, axioms in KB and propositional clause must be in definite clause,…

COMPLETE EVALUATIONS

CRITERIA	POINTS	EVALUATE	SUMMARY
Read the input data and successfully store it in some data structures.	1.0 pt	1.0/1.0 pt	100%
The output file strictly follows the lab specifications.	1.0 pt	1.0/1.0 pt	100%
Implement the propositional resolution algorithm.	2.0 pts	2.0/2.0 pts	100%
Provide a complete set of clauses and exact conclusion.	3.0 pts	3.0/3.0 pts	100%
Five test cases: both input and output files.	1.0 pt	1.0/1.0 pt	100%
Discussion on the algorithm's efficiency and suggestions.	2.0 pts	2.0/2.0 pts	100%
Done all.	10.0 pts	10.0/10.0 pts	100%

REFERENCES

[1]: Artificial Intelligence: A Modern Approach, 3rd edition – Stuart Russell, Peter Norvig.

[2]: Artificial Intelligence AIMA code, python code, logic section.

https://github.com/aimacode/aima-python/blob/master/logic.ipynb