|  |  |  |
| --- | --- | --- |
| ID | **18127259** | *Introduction to Artificial Intelligence*  *CSC14003*  Lab02: PL Resolution |
| Full name | Kiều Công Hậu |
| Class | 18CLC1 |

**REPORT**

1. **Check list:**

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Degree of completion |
| 1 | Read the input data and successfully store it in some data structures. | 100% |
| 2 | The output file strictly follows the lab specifications. | 100% |
| 3 | Implement the propositional resolution algorithm. | 100% |
| 4 | Provide a complete set of clauses and exact conclusion. | 100% |
| 5 | Five test cases: both input and output files. | 100% |
| 6 | Discussion on the algorithm’s efffiency and suggestions. | 100% |

1. **Breif description of main functions:**
2. ***main.py***
   1. ***main***

This function performs all of the following basic action with each test of 5 test cases:

* Read the input data and store it in appropriate data strutures.
* Call the function pl\_resolution, which implements the PL Resolution algorithm.
* Write the output data to the output file in valid format.

1. ***MyAlgorirthms.py***

Class MyAglorithms has 4 attributes:

* *alpha*: stores information of the alpha query (list).
* *KB*: stores information of the Knowledge Base (list).
* *new\_clauses\_list*: stores new clauses after each loop of the PL Resolution algorithm (list).
* *solution:* represent the result of the query (bool).
  1. ***read\_input\_data***

This function helps you read an input data from an input file into the Knowledge Base (*KB*) and Alpha (*alpha*).

All of clauses are standardized:

* Get rid of all of duplicated literals.
* Literals within a clause are sorted following the alphabetical order.

*KB* and *alpha* are standardized also:

* Get rid of all of duplicated clauses.
* Get rid of all of clauses in which two complementary literals appear.
  1. ***pl\_resolution***

This function helps you query *alpha* based on the *KB* by the PL Resolution algorithm, the result of the query *alpha* is stored in *solution*, all of new clauses of each loop are stored in *new\_clauses\_list*.

* 1. ***wirte\_output\_data***

This function helps you write an ouput data to an output file in valid format.

* 1. ***standard\_cnf\_sentence***

This function helps you to standardized a CNF sentence such as *KB* or *alpha*:

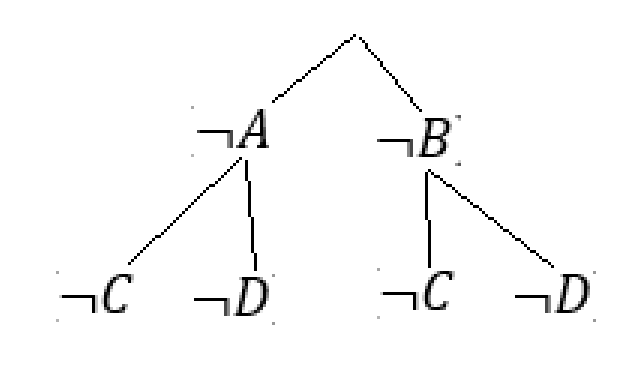
* Standardize all of clauses.
  + Get rid of all of duplicates.
  + Literals within a clause are sorted following the alphabetical order.
* Discard all of clauses in which 2 complementary literals appears.
  1. ***negation\_of\_cnf\_sentence***

This function helps you generate a negation of a CNF sentence. I use the distribution algorithm to implement this function.

Idea:

To implement the distribution, I create a tree recursively likes the below one. The result of the distribution is conjunction of clauses, with each clause is a disjunction of all literals on each branch of the tree.

Please read 2 functions generate\_combinations and generate\_combinations\_recursively for more understandings.



* 1. ***resolve***

This function helps you to resolve 2 clauses then return a list of resolvents (list of clauses).

1. **Discussion on the algorithm’s efficiency and suggestions to improve:**

According to the original PL Resolution algorithm, there are many redundant resolutions. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| KB and not alpha | Loop 1 | Loop 2 | Loop 3 |
| (1) … | (5) … | (8) … | (12) … |
| (2) … | (6) … | (9) … |  |
| (3) … | (7) … | (10) … |  |
| (4) … |  | (11) … |  |

At Loop 2, we don’t need to resolve (1) with (2), (1) with (3), … (3) with (4) because these resolution are done at Loop 1.

My suggestion: At loop n, we just need to resolve Ci with Cj (Ci is a clause from all of clauses in the above table, Cj is a clause from new clauses at loop n – 1).

For example: at loop 3, and .

**THE END**