Student ID: 18127208
Full name: Ngô Thanh Phương Thái

Lab02: PL Resolution

1. **Check list:**

Criteria	Evaluate
Read the input data and successfully store it in some data structures	**100%**
The output file strictly followsthe lab specifications	**100%**
Implement the propositional resolution algorithm	**100%**
Provide a complete set of clauses and exact conclusion	**100%**
Five test cases: both input and output files	**100%**
Discussion on the algorithm's efffiency and suggestions	**100%**

2. **Brief description of main functions:**

- **input_file, output_file**: Too basic so I am not going to explain here.
- **Knowledge**: Structure used to represent input, include KB and alpha.

```python
class Knowledge:
    def __init__(self):
        self.KB = []
        self.alpha = []
    def setKnowledge(self, KB, alpha):
        self.KB = KB
        self.alpha = alpha
    def getKB(self):
        return self.KB
    def getAlpha(self):
        return self.alpha
```

- **Library**: **copy** for **deepcopy** list, **product** of **itertools** for better loop!

```python
import copy
from itertools import product
from .knowledge import *
```

- **is_meaningless**: Function check clause is meaningless so it can be discard, check by if exist (**A != B and A == -B with A, B in clause**) then **True**.

```python
def is_meaningless(clause):
    is_suck = lambda i, j : i != j and is_negative(clause[i], clause[j])
    size = len(clause)
    return size == 0 or any(is_suck(i, j) for i, j in product(range(size), repeat=2)
```

- **is_stop**: Function check list clause is contain empty clause to end PL_resolution algorithm.

```
def is_stop(listClause):
    is_empty = lambda a, b : len(a) == 1 and len(b) == 1 and is_negative(a[0], b[0])
    return any(is_empty(a, b) for a, b in product(listClause, repeat=2))
```

- **PL_resolve**: Function return resolve of 2 clause, remove opposing literal pair and duplicate literals.

```
def PL_resolve(clauseA, clauseB):
    A = copy.deepcopy(clauseA)
    B = copy.deepcopy(clauseB)
    for iA in range(len(A)):
        for iB in range(len(B)):
            if is_negative(A[iA], B[iB]):
                A.pop(iA)
                B.pop(iB)
                return True, list(set(A + B))
    return False, A + B
```

- **new_list**: Function create new list clause from old list clause, new list will contain resolve of each pair of clause in old list if it can resolve, not meaningless and not in new list or old list.

```
def new_list(listClause):
    is_same = lambda clauseA, clauseB : sorted(clauseA) == sorted(clauseB)
    is_exist = lambda clause, listClause : any(is_same(tmp, clause) for tmp in listClause)

    res, count = [], 0
    for i in range(len(listClause) - 1):
        for j in range(i + 1, len(listClause)):
            req, clause = PL_resolve(listClause[i], listClause[j])
            if req and not is_meaningless(clause) and not is_exist(clause, listClause + res):
                res.append(clause)
    return res
```

- **negative_alpha**: Function return negative of alpha using **De Morgan** law and **Distribution** law

```python
def negative_alpha(alpha):
    negative = lambda x : x[1] if len(x) > 1 and x[0] == '-' else '-' + x
    distribution = lambda clauseA, clauseB : [a + b for a, b in product(clauseA, clauseB)]

    pre_neg = [[[negative(var)] for var in clause] for clause in alpha]
    res = pre_neg[0]
    for i in range(1, len(pre_neg)):
        res = distribution(res, pre_neg[i])

    return res
```

- **PL_resolution**: Function return sorted output of PL resolution follow pseudo code of **Lecture07-LogicalAgents page 53.**

```python
def PL_resolution(knowledge):
    sort_output = lambda output : [[sorted(clause, key=lambda x : x[1:] if x[0] == '-' else x) if type(clause) is not str else clause for clause in gr] if type(gr) is not str else gr for gr in output]

    output = []
    listClause = knowledge.getKB() + negative_alpha(knowledge.getAlpha())
    listNewClause = new_list(listClause)
    while (len(listNewClause) > 0):
        output.append(listNewClause)
        listClause += listNewClause
        listNewClause = new_list(listClause)
        if is_stop(listClause):
            listNewClause.append('{}')
            output += [listNewClause, 'YES']
            return sort_output(output)
    output += [[], 'NO']
    return sort_output(output)
```

- **main**: Main function will read **5 file** input in **inputs** folder and using PL resolution to check whether KB entails alpha then write output to **outputs** folder.

```python
from IO import *
from base.knowledge import *

if __name__ == '__main__':
    knowledge = Knowledge()
    for i in range(1, 6):
        input_file('inputs//input{}.txt'.format(i), knowledge)
        output_file('outputs//output{}.txt'.format(i), PL_resolution(knowledge))
```

3. Discussion on the algorithm's efffiency and suggestions:

- PL resolution is not optimize since many resolution steps are pointless and numbers of clauses (numbers of literals also) is very large, so deciding entailment might be done in **exponential time**.
- Suggestions: Use **Forward-chaining** or **Backward-chaning** instead, deciding entailment can be done in **linear time**.