

## 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 effciency 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.

```
1 class Knowledge:
2     def __init__(self):
3         self.KB = []
4         self.alpha = []
5     def setKnowledge(self, KB, alpha):
6         self.KB = KB
7         self.alpha = alpha
8     def getKB(self):
9         return self.KB
10    def getAlpha(self):
11        return self.alpha
```

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

```
1 import copy
2 from itertools import product
3 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.

```
1 def is_meaningless(clause):
2     is_suck = lambda i, j : i != j and is_negative(clause[i], clause[j])
3     size = len(clause)
4     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.

```

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

```

- **PL\_resolve**: Function return resolve of 2 clause, remove opposing symbol pair and duplicate symbols.

```

1 def PL_resolve(clauseA, clauseB):
2     A = copy.deepcopy(clauseA)
3     B = copy.deepcopy(clauseB)
4     for iA in range(len(A)):
5         for iB in range(len(B)):
6             if is_negative(A[iA], B[iB]):
7                 A.pop(iA)
8                 B.pop(iB)
9             return True, list(set(A + B))
10    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.

```

1 def new_list(listClause):
2     is_same = lambda clauseA, clauseB : sorted(clauseA) == sorted(clauseB)
3     is_exist = lambda clause, listClause : any(is_same(tmp, clause) for tmp in listClause)
4
5     res, count = [], 0
6     for i in range(len(listClause) - 1):
7         for j in range(i + 1, len(listClause)):
8             req, clause = PL_resolve(listClause[i], listClause[j])
9             if req and not is_meaningless(clause) and not is_exist(clause, listClause + res):
10                res.append(clause)
11    return res

```

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

```
1 def negative_alpha(alpha):
2     negative = lambda x : x[1] if len(x) > 1 and x[0] == '-' else '-' + x
3     distribution = lambda clauseA, clauseB : [a + b for a, b in product(clauseA, clauseB)]
4
5     pre_neg = [[negative(var)] for var in clause] for clause in alpha]
6     res = pre_neg[0]
7     for i in range(1, len(pre_neg)):
8         res = distribution(res, pre_neg[i])
9
10    return res
```

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

```
1 def PL_resolution(knowledge):
2     sort_output = lambda output : [[sorted(clause, key=lambda x : x[1:] if x[0] == '-' else x) if type(clause) is
3     not str else clause for clause in gr] if type(gr) is not str else gr for gr in output]
4
5     output = []
6     listClause = knowledge.getKB() + negative_alpha(knowledge.getAlpha())
7     listNewClause = new_list(listClause)
8     while (len(listNewClause) > 0):
9         output.append(listNewClause)
10        listClause += listNewClause
11        listNewClause = new_list(listClause)
12        if is_stop(listClause):
13            listNewClause.append('{}')
14            output += [listNewClause, 'YES']
15            return sort_output(output)
16        output += [[], 'NO']
17    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.

```
1 from IO import *
2 from base.knowledge import *
3
4 if __name__ == '__main__':
5     knowledge = Knowledge()
6     for i in range(1, 6):
7         input_file('inputs//input{}.txt'.format(i), knowledge)
8         output_file('outputs//output{}.txt'.format(i), PL_resolution(knowledge))
```

### 3. Discussion on the algorithm's efficiency and suggestions:

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