Name: Trần Đại Chí

Class: 18CLC3

Student ID: 18127070

1. Input/Ouput:
   - 5 input files for test case is putted in folder INPUT
   - Result of these input files is also putted in folder OUTPUT

2. Compile and run the program: just open src.py in folder SOURCE and run it, no more packages are required

3. Evaluate completion: 100% completed

4. Structure file src.py:

   + def readFile(myFile): read the input file. we'll read first line to get number clause query alpha, then read second line for clause in query alpha, add it to list and also remove character '\n' in list. Finally, read third line for number clause KB and read to end of file for all clause in KB, also add it to list and remove character 'OR' in list

   + def merge(ci, cj): merge pair of clauses Ci, Cj into a clause that call merge_clause

   + def NegativeCharacter(character): check if a list exist a negative element, example our list is: ['A', '-B'] -> '-B' is a negative element in our list

   + def isPointless(ci, cj): check if our list has a negative element of that element, example: ['A', 'B', '-A'] -> '-A' is a negative element of element 'A'

   + def Empty(clause): check if a clause is empty

   + def unique(list1): get unique values from a list, example our list is [1,2,1,1,3,4,3,3,5] -> list contains all unique values: [1,2,3,4,5]

   + def PL_Resolve(ci, cj): first, merge pair of clauses Ci, Cj into a new_clause, then we use 2 loop to iterate through ci and cj, example if our clause is ['B', 'C', 'C'] -> x = y (x in ci and y in cj) -> remove one element 'C' from list because if first C = second C = true/false -> first C ∧ second C or first C v second C also has result True/False, so it isn't necessary to have two element 'C' in this case. Else, example if our clause is ['-A', 'A', '-B'] -> remove both element '-A' and 'A' because -A v A = True that lead to -B v True -> it's pointless, so we just keep '-B' in our clause and remove both '-A' and 'A'

   + def checkPointlessMerge(merge_clause): after merging Ci, Cj into merge_clause, if merge_clause is pointless, return True and start resolving

   + def convert_negate(s): first, we get alpha, then check if index 0 of character alpha is positive, make it become negative by adding it with a character '-', else if index 0 of character alpha is negative, make it become positive by getting character alpha from index 1

   + def PL_Resolution(KB, alpha, f): first, we declare a list clauses = KB, but during resolving, clauses can change, so making it copy from KB to avoid some unexpected results, then use function convert_negate to convert alpha and add alpha after converting to list clauses. Next, we'll have a count variable for counting number of clauses generated in loop and create an empty list for storing new_clause. For each pair of clauses in Ci, Cj in clauses, we'll check if our merge_clause suc as ['-A', 'A', '-B'] which is pointless, start resolving it and also sort by alphabet after resolving. After generating all new sentences from KB and we can't generate more sentences, increase variable count and write new_clause to file. We also check pair of clause Ci, Cj after resolving is not pointless and it not exists in both list clauses and list new_clause, we'll add it to new_clause. In case our new_clause is empty and can't find any more empty clause, print character '0' and 'NO' because in this situation, KB doesn't entail by alpha

   + Bellow is the pseudo code for propositional logic resolution algorithm

```
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 $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  loop 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 $false$
     $clauses \leftarrow clauses \cup new$
```

Figure 7.12 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.